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»

Факультет ИТР

Кафедра ПИн

КУРСОВАЯ РАБОТА

По Теории автоматов и формальных языков

Тема Транслятор с подмножества языка Pascal

Руководитель

Кульков Я.Ю.
(фамилия, инициалы)

(подпись) (дата)

Студент ПИН - 122
(группа)

Буланов Е.А.
(фамилия, инициалы)

(подпись) (дата)

Муром 2024

Эта курсовая работа посвящена разработке транслятора с подмножества языка Pascal. В процессе разработки будут рассмотрены основные конструкции и особенности синтаксиса Pascal, а также все основные алгоритмы, такие как лексический, синтаксический анализаторы, разбор сложных выражений.

Проект включает в себя 34 страницы, 43 литературных источника, 2 таблицы и 9 приложений.

Работа состоит из введения, шести параграфов и заключения.

This course work is devoted to the development of a translator from a subset of the Visual Basic language. During the development process, the basic constructions and syntactic features of Pascal will be considered, as well as all the basic algorithms such as lexical, syntactic analysis and parsing of complex expressions.

The project contains 34 pages, 43 literary sources, 2 tables and 9 appendices.

The work consists of an introduction, six paragraphs and a conclusion.

Содержание

Введение

В текущей курсовой работе рассматривается разработка транслятора с подмножества языка Pascal, что является важной задачей в области теории автоматов и формальных языков.

Актуальность работы обусловлена необходимостью эффективных инструментов для автоматизации процесса разработки ПО. Транслятор с подмножества языка Pascal может быть полезен для разработчиков, так как позволяет переводить исходный код на этом языке программирования в другие форматы или языки.

Цель работы: Изучение основных алгоритмов и методов , применяемых в разработке трансляторов с подмножества формальных языков. На основе изученного материала разработать собственный транслятор с подмножества языка Pascal.

Задачи работы:

- Изучение основных принципов построения трансляторов
- Анализ лексической и синтаксической составляющих языка Pascal
- Изучение разновидностей грамматик формальных языков
- Рассмотреть различия методов трансляции арифметических и логических выражений, а также применить один из этих метовод в разработке собственного транслятора

Объектом изучения является процесс разработки транслятора с подмножества языка Pascal. Предметом изучения являются методы и алгоритмы, используемые при создании трансляторов, особенности синтаксиса и лексики языка Pascal.

Теоретическая важность данной работы заключается в применении знаний из области теории автоматов и формальных языков для решения конкретной практической задачи - разработки транслятора. Практическая значимость работы заключается в создании инструмента, потенциально используемого разработчиками для упрощения и ускорения процесса перевода исходного кода на языке Pascal.

1. Анализ технического задания

Задание – реализовать транслятор с подмножества некоторого языка программирования

1. Язык для трансляции – Pascal
2. Обеспечить развернутую диагностику ошибок
3. Реализовать класс транслятора
4. Синтаксический разбор – на основе LR – грамматик
5. Разбор выражений выполнять методом Дейкстри
6. В языке поддерживаются:
   - у идентификатора 8 символов значащие;
   - не менее 3-х директив описания переменных;
   - простой арифметический оператор;
   - сложное логическое выражение;
   - условный оператор if … then…else
7. Пример программы на заданном языке:

```
var a,b,c: integer;
begin
 a:=3; b := a;
 if ((a > b) and (b > 0)) then c:= a; else begin
 c := b; b := 1;
end;
end.
```

Решение задания выполняется в парадигме ООП, язык программирования – C#. Проект, состоящий из рабочего окна и программы, реализующей транслятор, представляет из себя приложение Windows Forms. В открывающемся окне есть четыре текстовых поля: одно, большое, для ввода текста программы на языке Pascal, остальные три для вывода лексем языка, ошибок, и обратной польской нотации вместе с матрицей арифметического оператора. В окне также имеются две кнопки, одна для открытия нужного файла и вторая, запуска транслятора.

Транслятор состоит из двух главных частей:

1. Лексический анализ входного кода
2. Синтаксический анализ

На этапе лексического анализа программа собирает все допустимые лексемы языка, которые могут в нём присутствовать. После этого синтаксический анализатор, реализованный на основе LR – грамматики, проверяет порядок

описания конструкций языка. В анализаторе также реализован разбор логических выражений методом Дейкстры. Если программа на любом из этапов зафиксировала определённую ошибку, то пользователю выдаётся сообщение с этой ошибкой.

2. Описание грамматики языка

Язык Pascal является компилируемым языком программирования, поддерживающим процедурное программирование и в некоторых версиях объектно-ориентированное программирование. Он был создан Никлаусом Виртом в 1970 году и предназначен для обучения программированию, а также для разработки различных приложений. Pascal активно используется в образовательных учреждениях и для разработки программного обеспечения, особенно в его объектно-ориентированном варианте Delphi.

Структура программы:

- Объявление трёх переменных a, b, c типа данных integer
- Два присваивания a = 3 и b = a
- Условный оператор if с условием (a > b) and (b > 0)
- В теле условия просиходит присваивание c = a или при не выполнении условия присваивается c = b и b = 1
- После чего программа завершается

Внутри условного оператора может находится вложенный условный оператор. Пример см. ниже.

Листинг кода №1 – Программа с вложенным условным оператором

```
var a,b,c: integer;
begin
 a:=3; b := a;
 if ((a > b) and (b > 0)) then c:= a; else begin
 c := b; b := 1; if ((a > b) and (b > 0)) then c:= a; else begin
 c := b; b := 1;
end;end;
end.
```

Грамматика языка

Состоит из множества терминалов и нетерминалов

G=[T,N,P, <прог>]

Множество терминалов
T={var, integer, real, longint, begin, end, if, then, else, id, lit,expr, noterm, +, -, *, \, ;, ,, :, =, ., >, <}
Множество нетерминалов
N{<прог>, <список>, <тело>, <опис>, <список перем>, <тип>, <список опер>, <опер>, <операнд>, , <усл>, }
Полученная грамматика
P{
<прог>::=var<список><тело>
<список>::=<опис>|<список><опис>
<опис>::=<список перем>:<тип>;
<тип>::=integer|real|longint
<список перем>::=id|<список перем>, id
<тело>::=begin<список опер>end.
<список опер>::=<опер>|<список опер><опер>
<опер>::=id:=<операнд>;|id:=<операнд><операнд>;|<усл>
<операнд>::=id|lit ::=+|*|-|\
<усл>::=if(expr)then<блок опер>| if(expr)then<блок опер>else<блок опер>
<блок опер>::=<опер>|begin<список опер>end;
}
Данная грамматика является контекстно-свободной грамматикой

3. Разработка архитектуры системы и алгоритмов

3.1. Архитектура системы

1. Интерфейс пользователя (Form):

- Текстовое поле для ввода программы на языке Pascal.

- три текстовых поля для вывода списка токенов, ошибок и матрицы лгоческого оператора вместе с обратной польской нотацией.

- Кнопка "открыть файл" для открытия программы из текстового файла.

- Кнопка "запустить анализатор" для запуска процесса трансляции и анализа программы.

2. Модуль лексического анализа:

- Функции для разделения входной программы на токены.

- Механизм для определения типа каждого токена (идентификатор, ключевое слово, оператор и т. д.).

- Функция для вывода токенов в соответствующее текстовое поле на форме.

3. Модуль синтаксического анализа:

- Алгоритмы для построения синтаксического дерева программы.

- Механизм для анализа структуры программы и выявления ошибок.

-Функция для вывода матрицы и обратной польской нотации логического оператора в соответствующее текстовое поле на форме.

4. Главный модуль приложения:

- Инициализация интерфейса пользователя.

- Обработчики событий кнопок "Открыть" и "Запустить".

- Взаимодействие с модулями лексического анализа, синтаксического анализа и выполнения программы.

3.2. Описание базовых алгоритмов

Алгоритм лексического анализа:

1. Из входного потока выбирается очередной символ, в зависимости от которого запускается тот или иной сканер (символ может быть также проигнорирован, либо признан ошибочным)
2. Просматривается входной поток символов программы на исходном языке, выделяя символы, входящие в требуемую лексему, до обнаружения очередного символа, который может ограничивать лексему, либо до обнаружения ошибочного символа
3. При успешном распознавании информация о выделенной лексеме заносится в список токенов. Алгоритм возвращается к первому этапу и продолжает просматривать входной поток с того места, на котором остановился сканер

Алгоритм построения решающей таблицы:

1. Берётся исходная грамматика языка, по ней строится граф состояний. Для его постройки берется исходное состояние и ставится маркер перед первым терминалом или нетерминалом. Если это терминал: то он раскрывается ниже и с ним посторяется тоже, что и с начальным состоянием. Если нетерминал: то он не раскрывается.
2. Если после маркера есть терминал или нетерминал, то пишется переход на следующее незанятое состояние, иначе пишется, что прехода нет. Первый и второй пункты продолжаются, пока не закончится грамматика.
3. Дальше, по графу состояний стротися решающая таблица. Берется нулевое состояние и у порождающего символа пишется «конец разбора» а у ε «сдвиг». Дальше рассматриваются поочереди все состояния из графа состояний, у символов слева от маркера пишется сдвиг, а у тех что справа состояние, в которое они переводят. В случае отсутсвия перехода пишется свертка с казанием количества сворачиваемых символов и нетерминал к которому переходит свертка .

На основе данных действий построины граф состояний (*см. Таблица №1, приложение 2*) и решающая таблица (*см. Таблица №2, приложение 2*).

Алгоритм синтаксического анализа:

1. Согласно решающей таблицы, строится каждое состояние с указанием метода сдвига и переходом к следующему состоянию. Выбор дейтсвия осужствляется текущим токеном в стеке.
2. Последовательно осуществляется проход по стеку токенов входной программы, и согласно правилам таблицы определяется действие(переход в следующее правило, либо сдвиг). Если текущая лексема не соответствует терминалу иил нетерминалу в правиле, фиксируется ошибка и разбор заканчивается

   Метод Дейкстры для разбора сложных выражений.

Метод Дейкстры для разбора сложных выражений. В методе используется один стек и выходная строка для хранения результата в обратной польской записи

(ОПЗ). Стек используется для хранения операторов и скобок. Выходная строка служит для записи операндов и операторов в порядке их обработки.

Алгоритм метода Дейкстры:

1. Просматриваем входную строку слева направо.
2. Если текущий элемент – операнд, добавляем его в выходную строку.
3. Если текущий элемент – операция, выполняем следующие действия:
   Извлекаем верхний элемент из стека.
   • Если стек пуст или элемент на вершине стека имеет меньший приоритет, чем текущая операция, помещаем текущую операцию в стек.
   • Если элемент на вершине стека имеет больший или равный приоритет, извлекаем элементы из стека и добавляем их в выходную строку, пока не найдем элемент с меньшим приоритетом или стек не опустеет, затем помещаем текущую операцию в стек.
4. Если текущий элемент – открывающая скобка, помещаем её в стек.
5. Если текущий элемент – закрывающая скобка, извлекаем элементы из стека и добавляем их в выходную строку до тех пор, пока не встретим открывающую скобку. Удаляем открывающую скобку из стека.

Возможные действия при обработке оператора OP из входной строки и элемента OP1 на вершине стека:

• D1 - Поместить OP в стек и читать следующий символ строки.

• D2 - Удалить OP1 из стека и добавить его в выходную строку; затем поместить OP в стек и читать следующий символ строки.

• D3 - Удалить OP1 из стека и читать следующий символ строки.

• D4 - Удалить OP1 из стека и добавить его в выходную строку.

• D5 - Ошибка в выражении. Конец разбора.

• D6 - Успешное завершение разбора.

После обработки всех символов входной строки извлекаем оставшиеся элементы из стека и добавляем их в выходную строку.

Входными данными для алгоритма являются текст исходной программы на языке программирования. Выходными данными является выражение в обратной польской записи (ОПЗ) или сообщение об ошибке в случае некорректного выражения.

4. Тестирование

Листинг кода №2

```
var a,b,c: integer;
begin
 a:=3; b := a;
 if ((a > b) and (b > 0)) then c:= a; else begin
 c := b; b := 1;
end;
end.
```

Данный фрагмент программы является корректным согласно правилам грамматики, и при его трансляции не ожидается никаких ошибок. При разборе этой программы задействованы алгоритмы лексического, синтаксического анализаторов, а также алгоритм разбора логического выражения методом Дейкстры. Результаты представлены на рисунке №2, приложение 3.

Листинг кода №3

```
var a,b,c: integer;
begin
 a:=3; b := a;
 if ((a > b) and (b > 0)) then c:= a; else begin
 c := b; b := 1; if ((a > b) and (b > 0)) then c:= a; else begin
 c := b; b := 1;
end;end;
end.
```

В данном примере представлен случай, когда в условном операторе if имеется ещё условный оператор. Так как данная программа не нарушает правил грамматики, то ожидается выполнение трансляции без ошибок. Результаты представлены на рисунке №3, приложение 3.

Листинг кода №4

```
Var 111a,b,c: integer;
begin
 a:=3; b := a;
 if ((a > b) and (b > 0)) then c:= a; else begin
 c := b; b := 1;
```

```
end;
end.
```

Начнём рассматривать примеры программ с ошибками. В данном фрагменте кода содержится лексическая ошибка, а именно название переменной, начинающееся с цифры. Содержится в строке “`Var 111a,b,c: integer;`”. Ожидается проверка работы алгоритма лексического анализатора, и дальнейшее завершение работы транслятора. Результат представлен на рисунке №4, приложение 3.

Листинг кода №5

```
var a,b,c: integer;
begin
 a:=3 b := a;
 if ((a > b) and (b > 0)) then c:= a; else begin
 c := b; b := 1;
end;
end.
```

В данном примере содержится одна синтаксическая ошибка, ошибка в присваивании переменной какого-либо значения. Содержится в строке “`a:=3 b := a;`” Согласно правилу грамматики “<опер>::=id:=<операнд>;”, после операнда должена следовать «;», а так как она отсутствует в программе, то синтаксический анализатор, зафиксировав данную ошибку, сообщает о ней. Результат представлен на рисунке №5, приложение 3.

Листинг кода №6

```
var a,b,c integer;
begin
 a:=3; b := a;
 if ((a > b) and (b > 0)) then c:= a; else begin
 c := b; b := 1;
end;
end.
```

Посмотрим на ещё один пример программы с синтаксической ошибкой. Содержится она в строке «`var a,b,c integer;`». Согласно правилу грамматики <опис>::=<список перем>:<тип>;, после списка переменных должно стоять двоеточие, но её нет. Синтаксический анализатор, зафиксировав данную ошибку, сообщает о ней. Результат представлен на рисунке №6, приложение 3.

Листинг кода №7

```
var a,b,c: integer;
begin
 a:=3; b := a;
 if ((a > b and (b > 0)) then c:= a; else begin
 c := b; b := 1;
end;
end.
```

В данном фрагменте программы содержится ошибка в логическом выражении, в строке «`if ((a > b and (b > 0)) then c:= a; else begin`». Данный пример проверяет корректность работы алгоритма разбора выражения. Так как в сравнении a и b нет закрывающей скобки, алгоритм фиксирует эту ошибку и сообщает о ней. Результат представлен на рисунке №7, приложение 3.

В данном разделе были приведены примеры корректных программ, и программ, содержащих ошибки.

5. Руководство пользователя

Данный продукт предназначен для обработки программы с подмножества языка Pascal. Обработка программы включает в себя её лексический и синтаксический анализ.

Приложение состоит из графического интерфейса, включающего в себя все необходимые компоненты. На окне располагаются две кнопки и четыре текстовых поля. Кнопка “открыть файл” позволяет пользователю выбрать любой текстовый файл, и занести его содержимое на главное текстовое поле, предназначенное для текста исходной программы на языке Pascal. Кнопка “запустить анализатор” запускает процесс лексического и синтаксического анализаторов, результаты работы которых выводятся на нижние текстовые поля. В левое поле выводятся все лексемы исходной программы, и их типы, в правое выводится матрица арифметического оператора и обратная польская нотация. В случае лексических и синтаксических ошибок в программе, они показываются в центральном нижнем текстовом поле.

Для начала работы с данным приложением необходимо ввести в главное текстовое поле исходную программу, или открыть её из текстового файла. После этого пользователь может отредактировать текст для соотвествия языку Pascal и запустить выполнение обработки данной программы с помощью соответствующей кнопки. Все результаты будут показаны на нижних текстовых полях.

Иллюстрация данного приложения представлена на рисунке №1, приложение 3.

6. Руководство программиста

Данный программный продукт предназначен для трансляции исходной программы на языке Pascal. Для выполнения данной цели программа должна выполнять лексический и синтаксический анализ.

Условия эксплуатации программы

- Наличие свободной оперативной памяти 10 МБ
- Операционная система Windows 7 и выше
- Процессор (программа не является требовательной к ресурсам процессора, подойдет любой оптимальный для ОС)
- Свободное место на диске: 5 МБ
- Устройства ввода: клавиатура, мышь
- Устройства вывода: монитор

Основные характеристики программы

Приложение разработано с помощью платформы .NET 8.0, в среде программирования Visual Studio 2022. AutomatTheory3.0Lexer.exe является исполняемым файлом программы. В проекте содержатся несколько файлов с исходным кодом на языке C#:

- Lexer.cs – содержит класс Lexer, реализующий лексический анализатор
- ParserLR.cs – содержит класс ParserLR, реализующий синтаксический анализатор
- Token.cs – содержит класс Token, описывающий тип и значение какого-либо токена(лексемы)
- TokenType.cs – содержит перечисление tokenType, в котором содержатся все типы токенов
- Form1.cs – содержит класс Form1, предназначенный для описания событий, происходящих на форме
- Expression.cs – содержит анкласс Expression, реализующий анализ сложных выражений

Назначение, структура входных и выходных данных программных функций

```
public class Lexer
```

Поля

```
private string code;
```

Содержит строку входной программы пользователя на языке Pascal

```
private List<string> lexerResult;
```

Результирующий список лексем, проанализированных лексическим анализатором. Представляет собой список строк.

```
public List<TokenType> types;
```

Результирующий список типов лексем, проанализированных лексическим анализатором.

```
static Dictionary<string, TokenType> specialWords;
```

Словарь, содержащий строковые представления специальных слов, и их тип токена

```
static Dictionary<char, TokenType> specialSymbols;
```

Словарь, содержащий специальные символы, и их тип токена

Методы

```
private static bool IsSpecialWord(string word)
```

Параметры

```
word string
```

Строка, проверяемая на то, является ли она специальным словом

Возвращаемое значение

```
bool
```

True – строка является специальным словом, иначе нет

```
private static bool IsSpecialSymbol(char symbol)
```

Параметры

```
char symbol
```

Символ, проверяемый на то, является ли он специальным

Возвращаемое значение

```
bool
```

True – символ является специальным, иначе нет

```
private void CreateTokenAndAddResult(TokenType type, string value)
```

Параметры

```
token Token
```

Токен, инициализирующийся в этом методе

```
type TokenType
```

Тип токена

```
value string
```

Строка, содержащая значение токена

Возвращаемое значение

```
void
private char IsDigitOrLeter(char c)
```

Параметры

```
c char
```

Символ, проверяемый на то, является ли он цифрой или буквой

Возвращаемое значение

```
char
```

Метод возвращает символ ‘d’, если параметр c является цифрой, ‘l’, если буквой, и само значение параметра, если он не является ни тем, ни другим

```
private void AddToken(string code)
```

Параметры

```
code string
```

Строка, содержащая какое-либо строковое представление значения токена

```
type TokenType
```

Тип токена

Возвращаемое значение

```
void
```

Исключения

```
Exception("встречен некорректный символ");
```

Метод принимает аргумент, содержащий значение null

```
public List<string> AnalyzeCode()
```

Параметры

```
string buffer = string.Empty;
```

Хранит символьную последовательность для слов

Возвращаемое значение

```
List<Token>
```

Список токенов, полученных в ходе анализа исходной программы

```
internal class ParserLR
```

Поля

```
List<Token> tokens = new List<Token>();
```

Список токенов исходной программы

```
Stack<Token> lexemStack = new Stack<Token>();
```

Стек токенов

```
Stack<int> stateStack = new Stack<int>();
```

Стек состояний

```
int nextLex = 0;
```

Переменная-счётчик следующей лексемы

```
int state = 0;
```

Переменная отображающая текущее состояния

```
bool isEnd = false;
```

Переменная окончания разбора

Методы

```
private void Shift()
```

Параметры

-

Возвращаемое значение

void

Назначение

Метод осуществляет переход к следующей лексеме, добавляя следюущюу с стек

```
private Token GetLexeme(int nextLexeme)
```

Параметры

```
int nextLexeme
```

Переменная обозначающая номер лексемы

Возвращаемое значение

```
Token
```

Токен

```
private void GoToState(int state)
```

Параметры

```
int state
```

Номер состояния

Возвращаемое значение

```
void
```

Метод перемещает в нужное состояние

```
private void Reduce(int number, string noterm)
```

Параметры

`int number`

Переменная обозначающая количество символов для свёртки

`string noterm`

Переменная обозначающая нетерминал в который сворачивается правило

Возвращаемое значение

`Void`

Метод сворачивает правило к начальному нетерминалу

Все остальные методы описывают правила грамматики, исходя из решающей таблицы, ни один из них не имеет параметров и возвращаемого значения. Пример одного из этих методов приведён ниже:

```
void State1()
```

Параметры

-

Возвращаемое значение

`void`

Метод описывает первое состояние из решающей таблицы(*см. Таблица №1*)

`class Token`

Свойства

```
public TokenType Type { get; set; }
```

Представляет собой тип токена

```
public string Value { get; set; }
```

Значение токена в строковом виде

Методы

```
public override string ToString()
```

Параметры

-

Возвращаемое значение

`string`

Возвращает строку формата “{Value} : {Type}”

`class TokenType`

Свойства

```
public enum tokenType
```

Содержит все типы токенов, имеющихся в подмножестве языка Visual Basic

```
public partial class Form1 : Form
```

Поля

-

Методы

```
public void ExprClear()
```

Парамаетры

```
string expr
```

Текст, передаваемый в метод

Возвращаемое значение

```
void
```

Метод выводит в специальное текстовое поле текст передаваемый из класса expression

```
public void ExprClear()
```

Параметры

-

Возвращаемое значение

```
void
```

Метод очищает текстовое поле для вывода из класса expression

```
private void ShowResult(List<Token> code)
```

Параметры

```
List<Token> code
```

Список токенов

Возвращаемое значение

```
void
```

Выводит список токенов в специальное поле

События

```
private void button2_Click(object sender, EventArgs e)
```

По нажатию кнопки “запустить анализатор” осуществляет запуск работы транслятора

```
private void button1_Click(object sender, EventArgs e)
```

При нажатии кнопки “открыть файл” появляется диалоговое окно проводника, из которого можно выбрать текстовый файл

Заключение

В результате данной работы был создан транслятор для подмножества языка Pascal. Данная программа выполняет лексический и синтаксический разбор, а также разбор сложного логического выражения. Для синтаксического анализа был использован метод LR-грамматик. Для разбора сложного логического выражения использовался метод Дейкстры.

В процессе работы была составлена грамматика подмножества языка Pascal, реализованы методы лексического и синтаксического разбора полученного кода, а также выполнено тестирование программы.

Подводя итоги, можно считать, что разработанный транслятор соответствует требованиям технического задания.

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. Пентус, А. Е. Математическая теория формальных языков : учебное пособие / А. Е. Пентус, М. Р. Пентус. — 3-е изд. — Москва : Интернет-Университет Информационных Технологий (ИНТУИТ), Ай Пи Ар Медиа, 2020. — 218 с.
3. 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.
4. Миронов, С. В. Формальные языки и грамматики : учебное пособие для студентов факультета компьютерных наук и информационных технологий / С. В. Миронов. — Саратов : Издательство Саратовского университета, 2019. — 80 с.
5. Малявко, А. А. Формальные языки и компиляторы : учебник / А. А. Малявко. — Новосибирск : Новосибирский государственный технический университет, 2014. — 431 с.

Приложение 1 - ссылка на репозиторий

Ссылка на репозиторий проекта: https://github.com/Dezmant8/AutomatTheoryTranslator.git

Приложение 2 - решающая таблица

состояние	предыдущее состояние	привала грамматики	переход
0		<прог>::=•var<список><тело>	1
1	0	<прог>::=var•<список><тело> <список>::=•<опис> <список>::=•<список><опис> <опис>::=•<список перем>:<тип>; <список перем>::=•id <список перем>::=•<список перем>, id	2 3 2 4 5 4
2	1	<прог>::=var<список>•<тело> <тело>::=•begin<список опер>end. <список>::=<список>•<опис> <опис>::=•<список перем>:<тип>; <список перем>::=•id <список перем>::=•<список перем>, id	6 7 8 4 5 4
3	1	<список>::=<опис>•	нет перехода
4	1, 2	<опис>::=<список перем>•:<тип>; <список перем>::=<список перем>•, id	9 10
5	1, 2	<список перем>::=id•	нет перехода
6	2	<прог>::=var<список><тело>•	нет перехода
7	2	<тело>::=begin•<список опер>end. <список опер>::=•<опер> <список опер>::=•<список опер><опер> <опер>::=•id:=<операнд>; <опер>::=•id:=<операнд><знак><операнд>; <опер>::=•id:=<усл>	11 12 11 13 13 13
8	2	<список>::=<список><опис>•	нет перехода
9	4	<опис>::=<список перем>:•<тип>; <тип>::=•integer <тип>::=•real <тип>::=•longint	14 15 16 17
10	4	<список перем>::=<список перем>, •id	18
11	7	<тело>::=begin<список опер>•end. <список опер>::=<список опер>•<опер> <опер>::=•id:=<операнд>; <опер>::=•id:=<операнд><знак><операнд>; <опер>::=•id:=<усл>	19 20 13 13 13

12	7, 43	<список опер>::=<опер>•	нет перехода
13	7, 12, 41, 43, 45,	<опер>::=id•:=<операнд>; <опер>::=id•:=<операнд><знак><операнд>; <опер>::=id•:=<усл>	21 21 21
14	9	<опис>::=<список перем>:<тип>•;	22
15	9	<тип>::=integer•	нет перехода
16	9	<тип>::=real•	нет перехода
17	9	<тип>::=longint•	нет перехода
18	10	<список перем>::=<список перем>, id•	нет перехода
19	11	<тело>::=begin<список опер>end•.	23
20	11	<список опер>::=<список опер><опер>•	нет перехода
21	13	<опер>::=id:•=<операнд>; <опер>::=id:•=<операнд><знак><операнд>; <опер>::=id:•=<усл>	24 24 24
22	14	<опис>::=<список перем>:<тип>;•	нет перехода
23	19	<тело>::=begin<список опер>end.•	нет перехода
24	21	<опер>::=id:=•<операнд>; <опер>::=id:=•<операнд><знак><операнд>; <операнд>::=•id <операнд>::=•lit <опер>::=id:=•<усл> <усл>::=•if(expr)then<блок опер> <усл>::=•if(expr)then<блок опер>else<блок опер>	25 25 26 27 28 29 29
25	24	<опер>::=id:=<операнд>•; <опер>::=id:=<операнд>•<знак><операнд>; <знак>::=•+ <знак>::=•* <знак>::=•- <знак>::=•\	30 31 32 33 34 35
26	24, 31	<операнд>::=id•	нет перехода

27	24, 31	<операнд>::=lit•	нет перехода
28	24	<опер>::=id:=<усл>•	нет перехода
29	24	<усл>::=if•(expr)then<блок опер> <усл>::=if•(expr)then<блок опер>else<блок опер>	36 36
30	25	<опер>::=id:=<операнд>;•	нет перехода
31	25	<опер>::=id:=<операнд><знак>•<операнд>; <операнд>::=•id <операнд>::=•lit	37 26 27
32	25	<знак>::=+•	нет перехода
33	25	<знак>::=*•	нет перехода
34	25	<знак>::=-•	нет перехода
35	25	<знак>::=\•	нет перехода
36	29	<усл>::=if(•expr)then<блок опер> <усл>::=if(•expr)then<блок опер>else<блок опер>	38 38
37	31	<опер>::=id:=<операнд><знак><операнд>•;	39
38	36	<усл>::=if(expr•)then<блок опер> <усл>::=if(expr•)then<блок опер>else<блок опер>	40 40
39	37	<опер>::=id:=<операнд><знак><операнд>;•	нет перехода
40	38	<усл>::=if(expr)•then<блок опер> <усл>::=if(expr)•then<блок опер>else<блок опер>	41 41
41	40	<усл>::=if(expr)then•<блок опер> <усл>::=if(expr)then•<блок опер>else<блок опер> <блок опер>::=•begin<список опер>end; <блок опер>::=•<опер> <опер>::=•id:=<операнд>; <опер>::=•id:=<операнд><знак><операнд>; <опер>::=•id:=<усл>	42 42 43 44 13 13 13
42	41	<усл>::=if(expr)then<блок опер>•	нет перехода

		<усл>::=if(expr)then<блок опер>•else<блок опер>	45
43	41, 45	<блок опер>::=begin•<список опер>end; <список опер>::=•<опер> <список опер>::=•<список опер><опер> <опер>::=•id:=<операнд>; <опер>::=•id:=<операнд><знак><операнд>; <опер>::=•id:=<усл>	46 12 11 13 13 13
44	41, 45	<блок опер>::=<опер>•	нет перехода
45	42	<усл>::=if(expr)then<блок опер>else•<блок опер> <блок опер>::=•begin<список опер>end; <блок опер>::=•<опер> <опер>::=•id:=<операнд>; <опер>::=•id:=<операнд><знак><операнд>; <опер>::=•id:=<усл>	47 43 44 13 13 13
46	43	<блок опер>::=begin<список опер>•end;	48
47	45	<усл>::=if(expr)then<блок опер>else<блок опер>•	нет перехода
48	46	<блок опер>::=begin<список опер>end•;	49
49	48	<блок опер>::=begin<список опер>end;•	нет перехода

Таблица №1 – граф состояний

состояние	стек разбора	вход	действие
0	<прог> ε var		конец разбора сдвиг s1
1	var <список> <опис> <список перем> id		сдвиг s2 s3 s4 s5
2	<список> <тело> begin <опис> <список перем> id		сдвиг s6 s7 s8 s4 s5
3	<опис>		свёртка(-1, <список>)
4	<список перем> : ,		сдвиг s9 s10

5	id		свёртка(-1, <список перем>)
6	<тело>		свёртка(-3, <прог>)
7	begin <список опер> <опер> id		сдвиг s11 s12 s13
8	<опис>		свёртка(-2, <список>)
9	: <тип> integer real longint		сдвиг s14 s15 s16 s17
10	, id		сдвиг s18
11	<список опер> end <опер> id		сдвиг s19 s20 s13
12	<опер>		свёртка(-1, <список опер>)
13	id :		сдвиг s21
14	<тип> ;		сдвиг s22
15	integer		свёртка(-1, <тип>)
16	real		свёртка(-1, <тип>)
17	longint		свёртка(-1, <тип>)
18	id		свёртка(-3, <список перем>)
19	end .		сдвиг s23
20	<опер>		свёртка(-2, <список опер>)
21	: =		сдвиг s24
22	;		свёртка(-4, <опис>)
23	.		свёртка(-4, <тело>)
24	= <операнд> id lit <усл> if		сдвиг s25 s26 s27 s28 s29
25	<операнд> ; <знак> + *		сдвиг s30 s31 s32 s33

	-		s34
	\		s35
26	id		свёртка(-1, <операнд>)
27	lit		свёртка(-1, <операнд>)
28	<усл>		свёртка(-4, <опер>)
29	if		сдвиг
	(		s36
30	;		свёртка(-5, <опер>)
31	<знак>		сдвиг
	<операнд>		s37
	id		s26
	lit		s27
32	+		свёртка(-1, <знак>)
33	*		свёртка(-1, <знак>)
34	-		свёртка(-1, <знак>)
35	\		свёртка(-1, <знак>)
36	(		сдвиг
	expr		s38
37	<операнд>		сдвиг
	;		s39
38	expr		сдвиг
	)		s40
39	;		свёртка(-7, <опер>)
40	)		сдвиг
	then		s41
41	then		сдвиг
	<блок опер>		s42
	begin		s43
	<опер>		s44
	id		s13
42	<блок опер>		свёртка(-6, <усл>)
	else		s45
43	begin		сдвиг
	<список опер>		s46
	<опер>		s12
	id		s13
44	<опер>		свёртка(-1, <блок опер>)
45	else		сдвиг
	<блок опер>		s47
	begin		s43
	<опер>		s44
	id		s13
46	<список опер>		сдвиг
	end		s48
47	<блок опер>		свёртка(-8, <усл>)

48	end ;		сдвиг s49
49	;		свёртка(-4, <блок опер>)

Таблица №2 – Решающая таблица

Приложение 3 - выводы программы

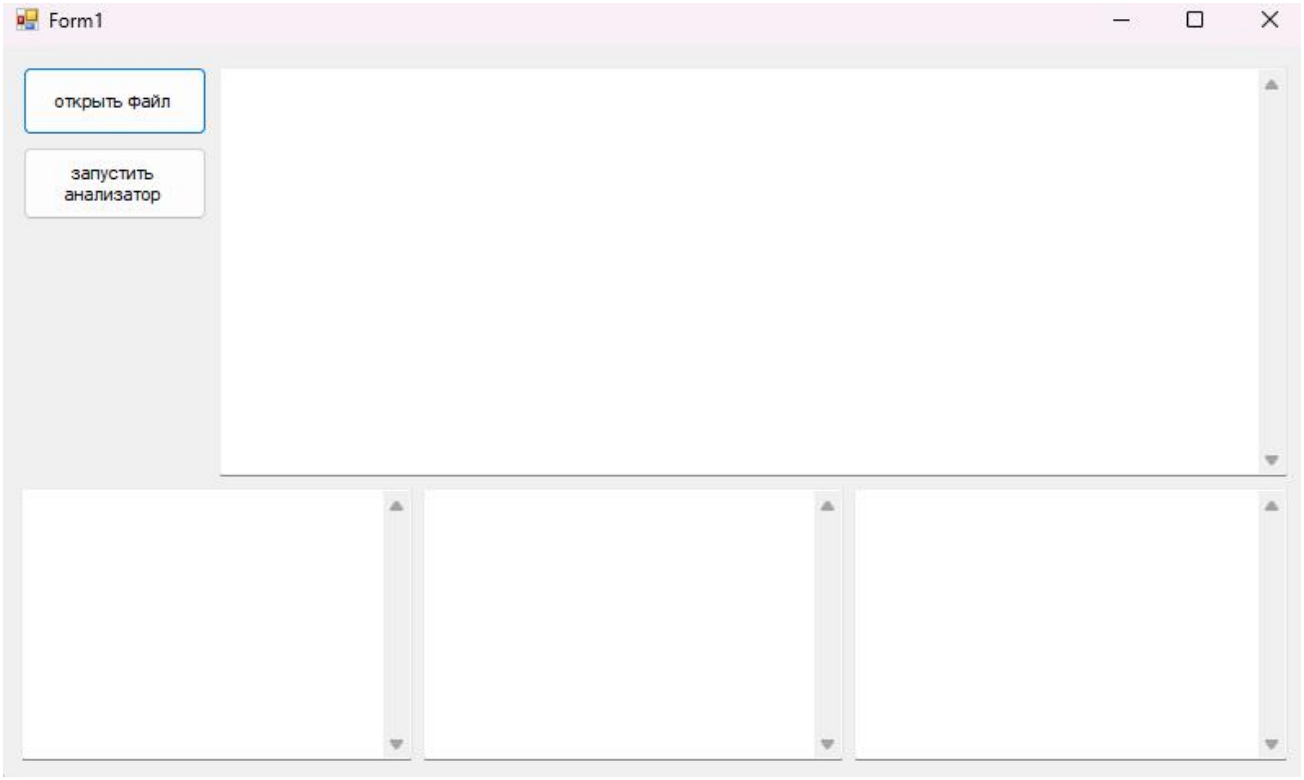

Рисунок №1 – Внешний вид приложения

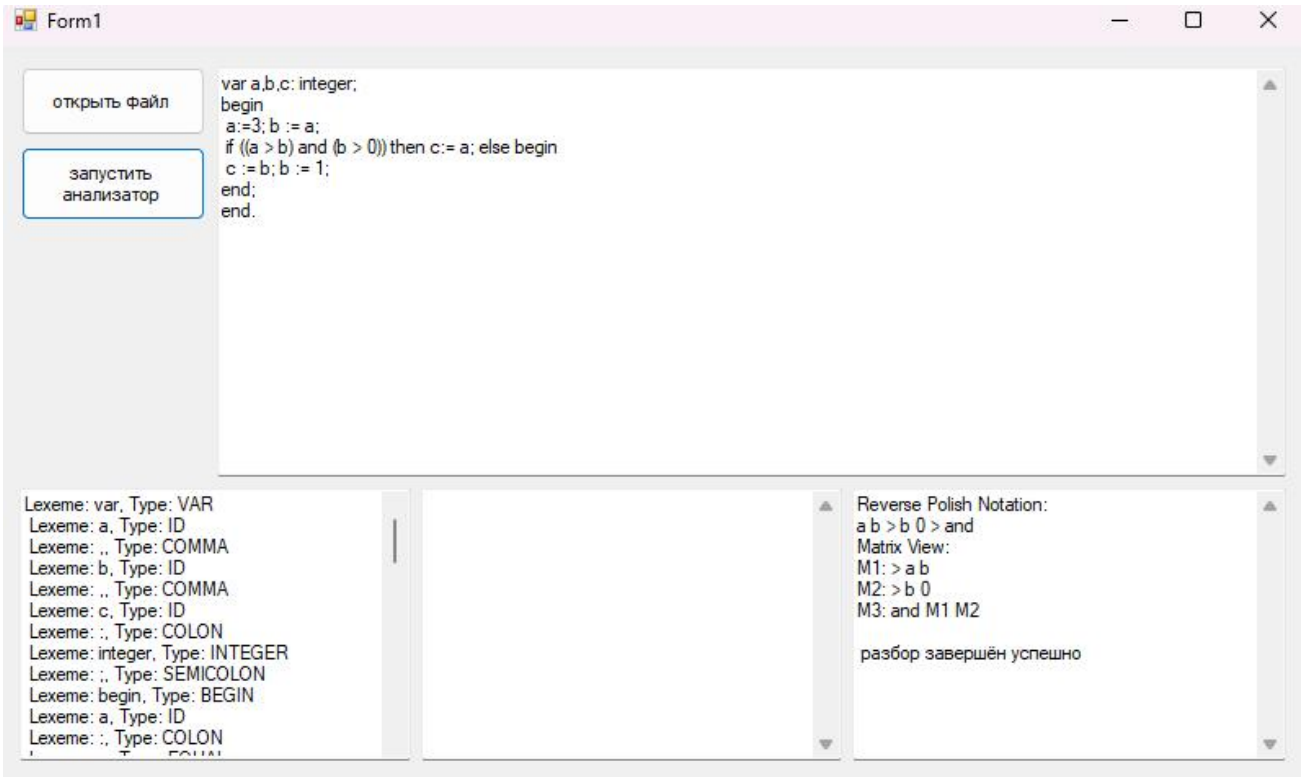

Рисунок №2 – Тестирование корректного фрагмента программы

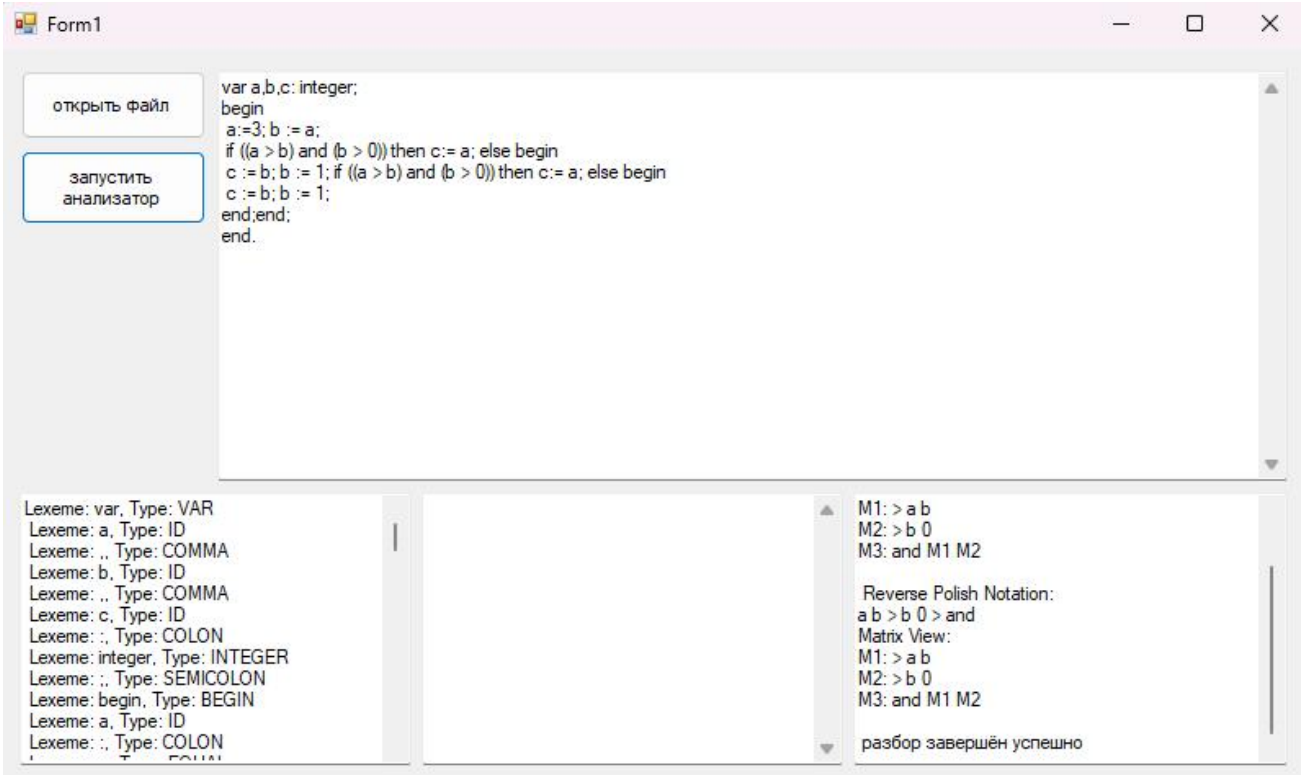

Рисунок №3 - Тестирование корректного фрагмента программы

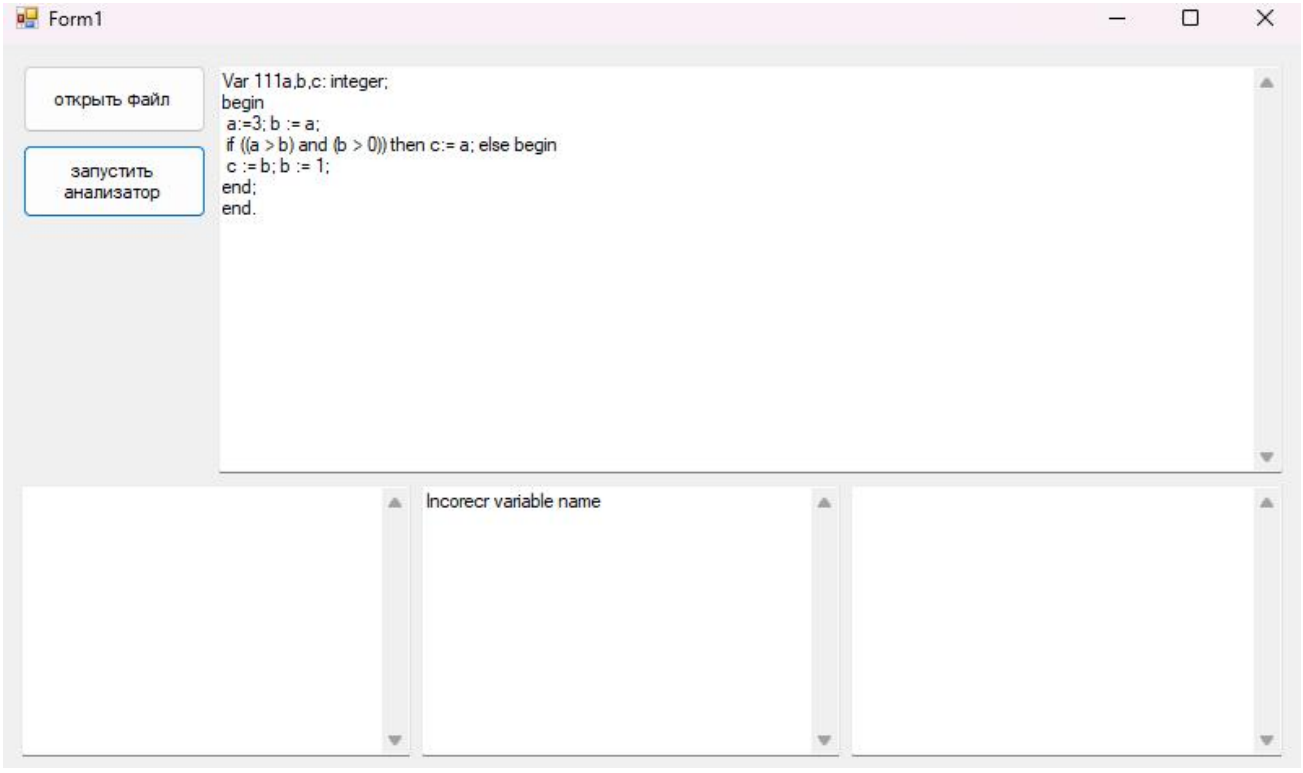

Рисунок №4 - Собщение о лексической ошибке

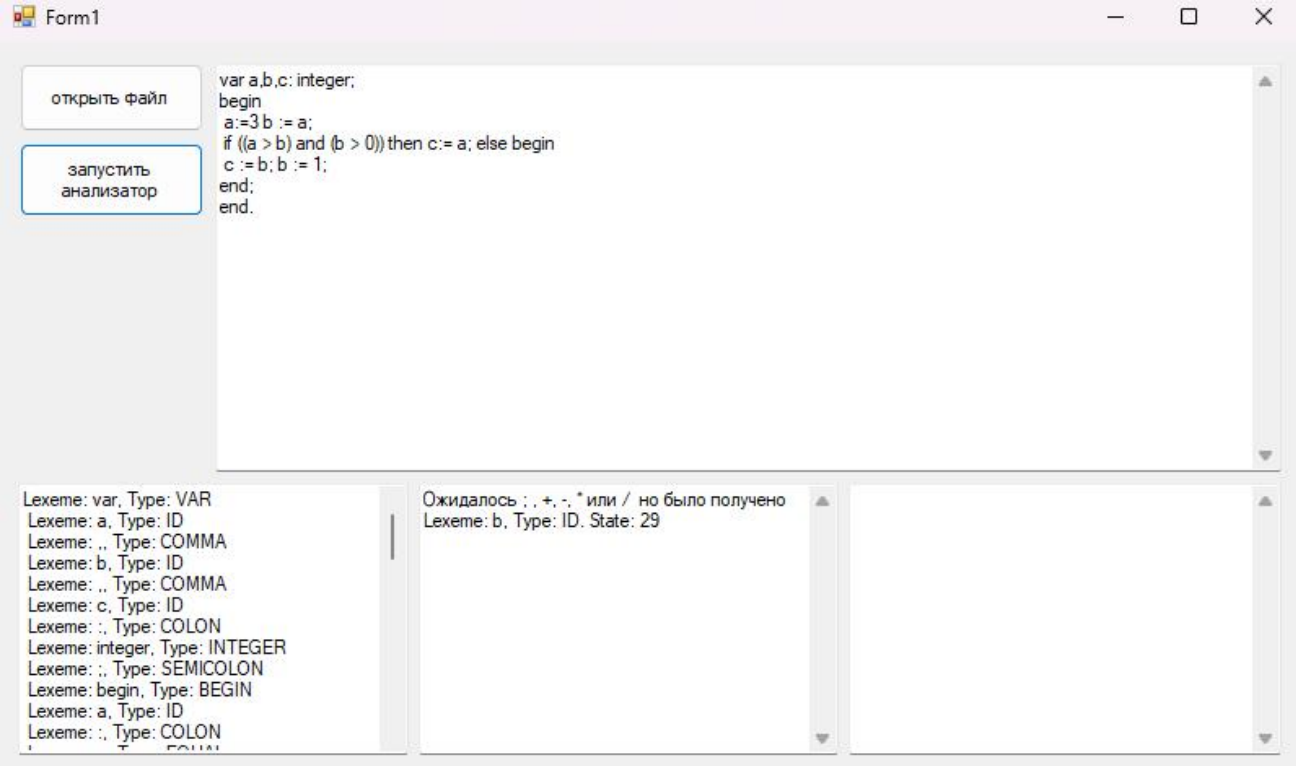

Рисунок №5 - Сообщение о синтаксической ошибке

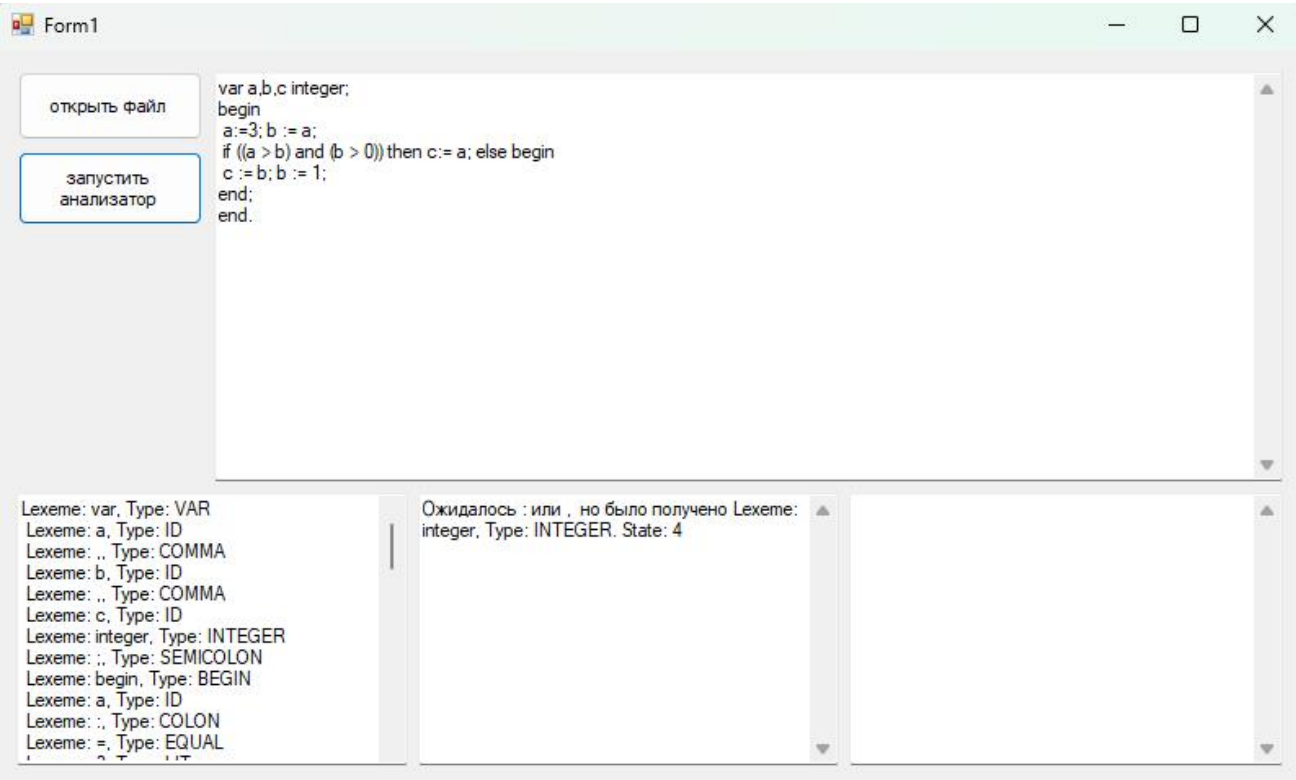

Рисунок №6 – Сообщение о синтаксической ошибке

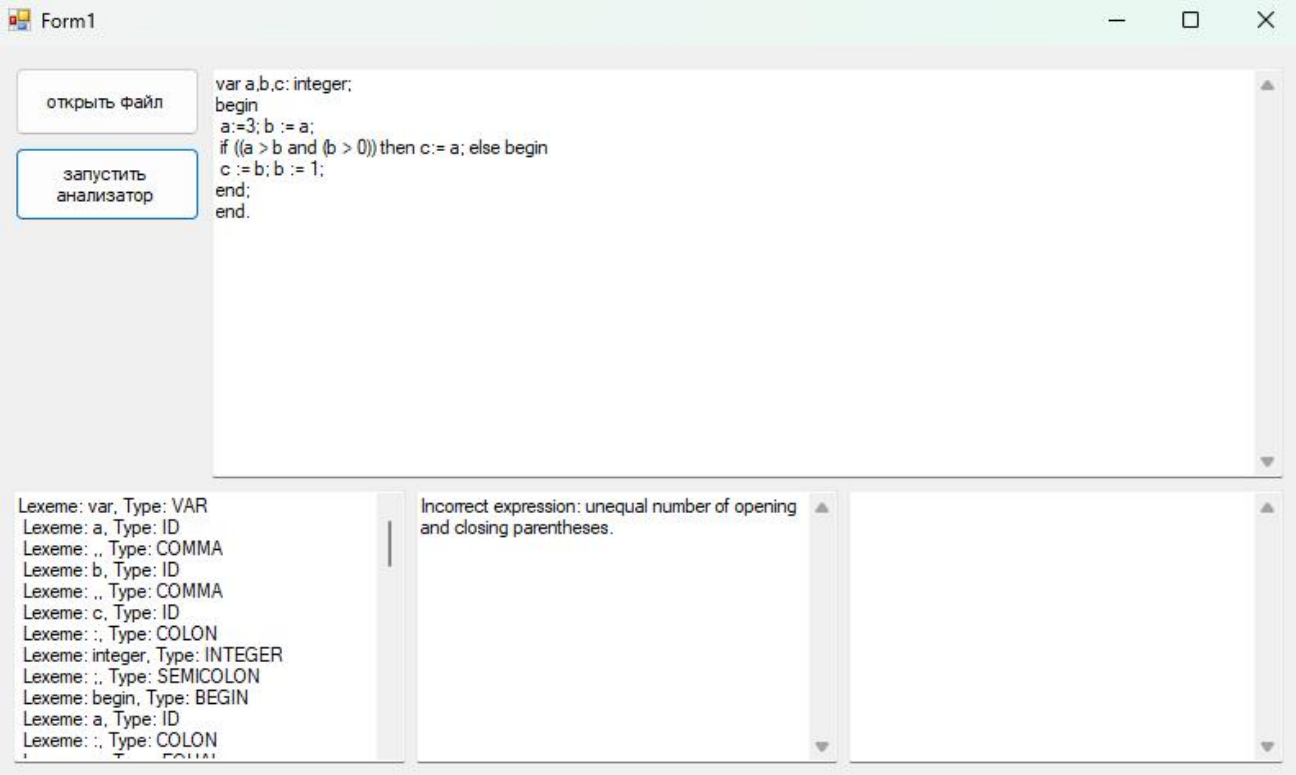

Рисунок №7 – Сообщение об ошибке в логическом выражении