МИНОБРНАУКИ РОССИИ

Федеральное государственное бюджетное образовательное учреждение высшего образования

«Ижевский государственный технический университет имени М. Т. Калашникова» (ФГБОУ ВО «ИжГТУ имени М.Т. Калашникова»)

Факультет «Информатика и вычислительная техника»

Кафедра «Автоматизированные системы обработки информации и управления»

ОТЧЁТ

По лабораторной работе №4

по дисциплине «Формальные языки и трансляторы»

на тему: «Разработка контекстного анализатора.»

Выполнил

Студент группы Б19-782-2 И.С. Пономарёв

Руководитель

К.т.н. доцент кафедры «АСОИУ» Д. Р. Касимов

г. Ижевск 2022

# Постановка задачи

Включить в программу синтаксического анализатора из лабораторной работы № 3 действия по вычислению атрибутов и проверки контекстных условий. Контекстное условие: все идентификаторы должны быть разными.

КС-грамматика:

D → K T

T → ∨ K T | ε

K → A E

E → ∧A E | ε

A → ¬ A | ( D ) | O

O → <1>=<2> | <2>=<1> | <2>=<2>

# Атрибутная грамматика обеспечивающая заданное контекстное условие

В таблице 2.1 показана атрибутная грамматика, обеспечивающая заданное контекстное условие, в которой s ↓ - ссылка на область памяти, где хранится список значений идентификаторов.

|  |  |
| --- | --- |
| Продукция | Семантические правила |
| D → K T | K. s ↓ = D. s ↓; T. s ↓ = D. s ↓ |
| T → ∨ K T1 | K. s ↓ = T. s ↓; T1. s ↓= T. s ↓ |
| K → A E | A. s ↓ = K. s ↓; E. s ↓=K. s ↓ |
| E → ∧A E1 | A. s ↓ = E. s ↓ ; E1. s ↓ = E. s ↓ |
| A → ¬ A1 | A1. s ↓=A. s ↓ |
| A → (D) | D. s ↓ = A. s ↓ |
| A → O | O. s ↓ = A. s ↓ |
| O → <1>=<2> | Условие: {<1> } ∉ O. s ↓  O. s ↓ .Add( <1> )  Условие: {<2> } ∉ O. s ↓  O. s ↓ .Add( <2> ) |
| O → <2>=<1> | Условие: {<2> } ∉ O. s ↓  O. s ↓ .Add( <2> )  Условие: {<1> } ∉ O. s ↓  O. s ↓ = O. s ↓ ⋃ { <1> } |
| O → <2>=<2> | Условие: {<2> } ∉ O. s ↓  O. s ↓ .Add( <2> )  Условие: {<2> } ∉ O. s ↓  O. s ↓ .Add( <2> ) |
|  | Для начального D s ↓= new List<string>(); |

# Код программы

using LAB4;

using System;

using System.Collections.Generic;

using System.Linq;

using System.Text;

using System.Threading.Tasks;

using System.Windows.Markup;

using System.Xml.Linq;

namespace LAB4

{

class ContextAnException : Exception

{

// Позиция возникновения исключительной ситуации в анализируемом тексте.

private int lineIndex; // Индекс строки.

private int symIndex; // Индекс символа.

// Индекс строки, где возникла исключительная ситуация - свойство только для чтения.

public int LineIndex

{

get { return lineIndex; }

}

// Индекс символа, на котором возникла исключительная ситуация - свойство только для чтения.

public int SymIndex

{

get { return symIndex; }

}

// Конструктор исключительной ситуации.

// message - описание исключительной ситуации.

// lineIndex и symIndex - позиция возникновения исключительной ситуации в анализируемом тексте.

public ContextAnException(string message, int lineIndex, int symIndex)

: base(message)

{

this.lineIndex = lineIndex;

this.symIndex = symIndex;

}

}

// Класс исключительных ситуаций синтаксического анализа.

class SynAnException : Exception

{

// Позиция возникновения исключительной ситуации в анализируемом тексте.

private int lineIndex; // Индекс строки.

private int symIndex; // Индекс символа.

// Индекс строки, где возникла исключительная ситуация - свойство только для чтения.

public int LineIndex

{

get { return lineIndex; }

}

// Индекс символа, на котором возникла исключительная ситуация - свойство только для чтения.

public int SymIndex

{

get { return symIndex; }

}

// Конструктор исключительной ситуации.

// message - описание исключительной ситуации.

// lineIndex и symIndex - позиция возникновения исключительной ситуации в анализируемом тексте.

public SynAnException(string message, int lineIndex, int symIndex)

: base(message)

{

this.lineIndex = lineIndex;

this.symIndex = symIndex;

}

}

// Класс "Синтаксический анализатор".

// При обнаружении ошибки в исходном тексте он генерирует исключительную ситуацию SynAnException или LexAnException.

class SyntaxAnalyzer

{

private LexicalAnalyzer lexAn; // Лексический анализатор.

// Конструктор синтаксического анализатора.

// В качестве параметра передается исходный текст.

public SyntaxAnalyzer(string inputLines)

{

// Создаем лексический анализатор.

// Передаем ему текст.

lexAn = new LexicalAnalyzer(inputLines);

}

// Обработать синтаксическую ошибку.

// msg - описание ошибки.

private void SyntaxError(string msg)

{

// Генерируем исключительную ситуацию, тем самым полностью прерывая процесс анализа текста.

throw new SynAnException(msg, lexAn.CurLineIndex, lexAn.CurSymIndex);

}

private void ContextError(string msg)

{

// Генерируем исключительную ситуацию, тем самым полностью прерывая процесс анализа текста.

throw new ContextAnException(msg, lexAn.CurLineIndex, lexAn.CurSymIndex);

}

// Проверить, что тип текущего распознанного токена совпадает с заданным.

// Если совпадает, то распознать следующий токен, иначе синтаксическая ошибка.

private void Match(TokenKind tkn)

{

if (lexAn.Token.Type == tkn) // Сравниваем.

{

lexAn.RecognizeNextToken(); // Распознаем следующий токен.

}

else

{

SyntaxError("Ожидалось " + tkn.ToString()); // Обнаружена синтаксическая ошибка.

}

}

// Проверить, что текущий распознанный токен совпадает с заданным (сравнение производится в нижнем регистре).

// Если совпадает, то распознать следующий токен, иначе синтаксическая ошибка.

private void Match(string tkn)

{

if (lexAn.Token.Value.ToLower() == tkn.ToLower()) // Сравниваем.

{

lexAn.RecognizeNextToken(); // Распознаем следующий токен.

}

else

{

SyntaxError("Ожидалось " + tkn); // Обнаружена синтаксическая ошибка.

}

}

// Процедура разбора для стартового нетерминала E.

private void D(out SyntaxTreeNode node,List<string> values)

{

node = new SyntaxTreeNode("D");

SyntaxTreeNode nodeK;

K(out nodeK, values);

node.AddSubNode(nodeK);

if (lexAn.Token.Type == TokenKind.LogicSum)

{

SyntaxTreeNode nodeT;

T(out nodeT,values); // Вызываем процедуру разбора для нетерминала T.

node.AddSubNode(nodeT);

}

}

private void K(out SyntaxTreeNode node, List<string> values)

{

node = new SyntaxTreeNode("K");

SyntaxTreeNode nodeA;

A(out nodeA,values);

node.AddSubNode(nodeA);

if (lexAn.Token.Type == TokenKind.LogicMultiply)

{

SyntaxTreeNode nodeE;

E(out nodeE,values);

node.AddSubNode(nodeE);

}

}

private void E(out SyntaxTreeNode node, List<string> values)

{

node = new SyntaxTreeNode("E");

node.AddSubNode(new SyntaxTreeNode(lexAn.Token.Value));

lexAn.RecognizeNextToken();

SyntaxTreeNode nodeA;

A(out nodeA,values);

node.AddSubNode(nodeA);

if (lexAn.Token.Type == TokenKind.LogicMultiply)

{

SyntaxTreeNode nodeE;

E(out nodeE,values);

node.AddSubNode(nodeE);

}

}

private void A(out SyntaxTreeNode node, List<string> values)

{

node = new SyntaxTreeNode("A");

switch (lexAn.Token.Type)

{

case TokenKind.Negative:

node.AddSubNode(new SyntaxTreeNode(lexAn.Token.Value));

lexAn.RecognizeNextToken();

SyntaxTreeNode nodeA;

A(out nodeA, values);

node.AddSubNode(nodeA);

break;

case TokenKind.LeftParenthesis:

node.AddSubNode(new SyntaxTreeNode(lexAn.Token.Value));

lexAn.RecognizeNextToken();

SyntaxTreeNode nodeD;

D(out nodeD,values);

node.AddSubNode(nodeD);

Match(TokenKind.RightParenthesis);

node.AddSubNode(new SyntaxTreeNode(")"));

break;

default:

SyntaxTreeNode nodeO;

O(out nodeO,values);

node.AddSubNode(nodeO);

break;

}

}

private void O(out SyntaxTreeNode node,List<string> values)

{

node = new SyntaxTreeNode("O");

switch (lexAn.Token.Type)

{

case TokenKind.FirstWord:

node.AddSubNode(new SyntaxTreeNode(lexAn.Token.Value));

if (values.Contains(lexAn.Token.Value)) SyntaxError($" Идентификатор {lexAn.Token.Value} уже встречался");

values.Add(lexAn.Token.Value);

lexAn.RecognizeNextToken();

Match(TokenKind.Equal);

node.AddSubNode(new SyntaxTreeNode("="));

var tokenValue = lexAn.Token.Value;

Match(TokenKind.SecondWord);

node.AddSubNode(new SyntaxTreeNode(tokenValue));

if (values.Contains(tokenValue)) SyntaxError($" Идентификатор {tokenValue} уже встречался");

values.Add(tokenValue);

break;

case TokenKind.SecondWord:

node.AddSubNode(new SyntaxTreeNode(lexAn.Token.Value));

if (values.Contains(lexAn.Token.Value)) SyntaxError($" Идентификатор {lexAn.Token.Value} уже встречался");

values.Add(lexAn.Token.Value);

lexAn.RecognizeNextToken();

Match(TokenKind.Equal);

node.AddSubNode(new SyntaxTreeNode("="));

if (lexAn.Token.Type != TokenKind.FirstWord

&& lexAn.Token.Type != TokenKind.SecondWord)

SyntaxError("Ожидалось слово типа 1 или типа 2");

node.AddSubNode(new SyntaxTreeNode(lexAn.Token.Value));

if (values.Contains(lexAn.Token.Value)) SyntaxError($" Идентификатор {lexAn.Token.Value} уже встречался");

values.Add(lexAn.Token.Value);

lexAn.RecognizeNextToken();

break;

default:

SyntaxError("Ожидалось слово типа 1 или 2");

break;

}

}

// Процедура разбора для нетерминала T.

private void T(out SyntaxTreeNode node,List<string> values)

{

node = new SyntaxTreeNode("T");

node.AddSubNode(new SyntaxTreeNode(lexAn.Token.Value));

lexAn.RecognizeNextToken();

SyntaxTreeNode nodeK;

K(out nodeK,values);

node.AddSubNode(nodeK);

if (lexAn.Token.Type == TokenKind.LogicSum)

{

SyntaxTreeNode nodeT;

T(out nodeT,values); // Вызываем процедуру разбора для нетерминала T.

node.AddSubNode(nodeT);

}

}

// Провести синтаксический анализ текста.

public void ParseText(out SyntaxTreeNode treeRoot)

{

lexAn.RecognizeNextToken(); // Распознаем первый токен в тексте.

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

D(out treeRoot,values); // Вызываем процедуру разбора для стартового нетерминала E.

if (!values.GroupBy(x => x).All(x => x.Count() == 1))

{

ContextError("Обнаружена контекстная ошибка: обнаружены повторяющиеся идентификаторы");

}

if (lexAn.Token.Type != TokenKind.EndOfText) // Если текущий токен не является концом текста.

{

SyntaxError("После арифметического выражения идет еще какой-то текст"); // Обнаружена синтаксическая ошибка.

}

}

}

}

# Пример работы программы

На рисунке 4.1 показан пример работы программы

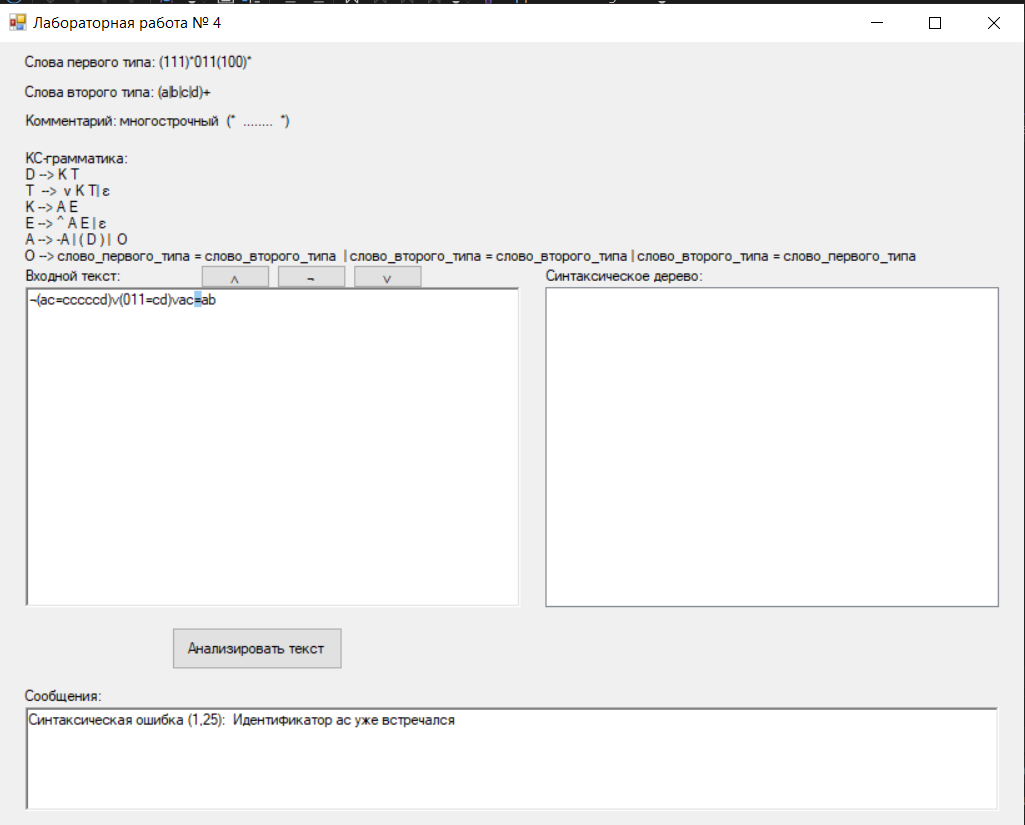


Рисунок 4.1 – Пример работы программы