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 №3

по дисциплине «Формальные языки и трансляторы»

на тему: «Построение синтаксического дерева.»

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 синтаксический анализатор из лабораторной работы № 2 построение синтаксического дерева. Использовать атрибутный метод Кнута, т.е. преобразовать КС–грамматику из лабораторной работы № 2 в атрибутную грамматику с добавлением атрибутов и правил построения синтаксического дерева.

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>

# Перевод КС-грамматики в атрибутную для построения синтаксического дерева

В таблице 2.1 показана атрибутная грамматика для построения синтаксического дерева, в которой функция mknode принимает переменное число аргументов (первый – метка создаваемого узла, остальные – указатели на дочерние узлы), функция mkleaf создает листьевой узел с меткой, указанной в параметре.

|  |  |
| --- | --- |
| Продукция | Семантические правила |
| D → K T | D.↑n:=mknode('D', K.↑n, T.↑n) |
| T → ∨ K T | T..↑n:=mknode(‘T’, mkleaf(‘∨'), K.↑n , T. ↑n) |
| K → A E | K.↑n:=mknode(‘K’, A.↑n, E.↑n) |
| E → ∧A E | E.↑n:=mknode(‘E’, mkleaf(‘∧’), A.↑n, E. ↑n) |
| A → ¬ A1 | A.↑n:=mknode(‘A’, mkleaf(‘¬’), A1.↑n) |
| A → (D) | A.↑n:=mknode(‘A’, mkleaf(‘(‘) , D.↑n , mkleaf(‘)’)) |
| A → O | A.↑n:=mknode(‘A’, O.↑n) |
| O → <1>=<2> | O.↑n:=mknode(‘O’, mkleaf(<1>), mkleaf(‘=’), mkleaf(<2>) |
| O → <2>=<1> | O.↑n:=mknode(‘O’, mkleaf(<2>), mkleaf(‘=’), mkleaf(<1>) |
| O → <2>=<2> | O.↑n:=mknode(‘O’, mkleaf(<2>), mkleaf(‘=’), mkleaf(<2>) |

# Код программы

Ниже представлен код программы.

using System;

using System.Collections.Generic;

using System.Linq;

using System.Text;

using System.Threading.Tasks;

using System.Xml.Linq;

namespace LAB3

{

// 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);

}

// 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)

{

node = new SyntaxTreeNode("D");

SyntaxTreeNode nodeK;

K(out nodeK);

node.AddSubNode(nodeK);

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

{

SyntaxTreeNode nodeT;

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

node.AddSubNode(nodeT);

}

}

private void K(out SyntaxTreeNode node)

{

node = new SyntaxTreeNode("K");

SyntaxTreeNode nodeA;

A(out nodeA);

node.AddSubNode(nodeA);

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

{

SyntaxTreeNode nodeE;

E(out nodeE);

node.AddSubNode(nodeE);

}

}

private void E(out SyntaxTreeNode node)

{

node = new SyntaxTreeNode("E");

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

lexAn.RecognizeNextToken();

SyntaxTreeNode nodeA;

A(out nodeA);

node.AddSubNode(nodeA);

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

{

SyntaxTreeNode nodeE;

E(out nodeE);

node.AddSubNode(nodeE);

}

}

private void A(out SyntaxTreeNode node)

{

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);

node.AddSubNode(nodeA);

break;

case TokenKind.LeftParenthesis:

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

lexAn.RecognizeNextToken();

SyntaxTreeNode nodeD;

D(out nodeD);

node.AddSubNode(nodeD);

Match(TokenKind.RightParenthesis);

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

break;

default:

SyntaxTreeNode nodeO;

O(out nodeO);

node.AddSubNode(nodeO);

break;

}

}

private void O(out SyntaxTreeNode node)

{

node = new SyntaxTreeNode("O");

switch (lexAn.Token.Type)

{

case TokenKind.FirstWord:

node.AddSubNode(new SyntaxTreeNode(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));

break;

case TokenKind.SecondWord:

node.AddSubNode(new SyntaxTreeNode(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));

lexAn.RecognizeNextToken();

break;

default:

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

break;

}

}

// Процедура разбора для нетерминала T.

private void T(out SyntaxTreeNode node)

{

node = new SyntaxTreeNode("T");

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

lexAn.RecognizeNextToken();

SyntaxTreeNode nodeK;

K(out nodeK);

node.AddSubNode(nodeK);

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

{

SyntaxTreeNode nodeT;

T(out nodeT); // 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.

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

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.

}

}

}

}

using System;

using System.Collections.Generic;

using System.Linq;

using System.Text;

using System.Threading.Tasks;

namespace LAB3

{

// Класс "Узел синтаксического дерева".

class SyntaxTreeNode

{

private string name; // Наименование узла.

private List<SyntaxTreeNode> subNodes; // Подчиненные узлы.

// Конструктор узла.

public SyntaxTreeNode(string name)

{

this.name = name;

subNodes = new List<SyntaxTreeNode>();

}

// Добавить узел в список подчиненных узлов.

public void AddSubNode(SyntaxTreeNode subNode)

{

this.subNodes.Add(subNode);

}

// Наименование узла - свойство только для чтения.

public string Name

{

get { return name; }

}

// Список подчиненных узлов - свойство только для чтения.

public List<SyntaxTreeNode> SubNodes

{

get { return subNodes; }

}

}

}

# Пример работы

На рисунке 4.1 показан пример работы программы.

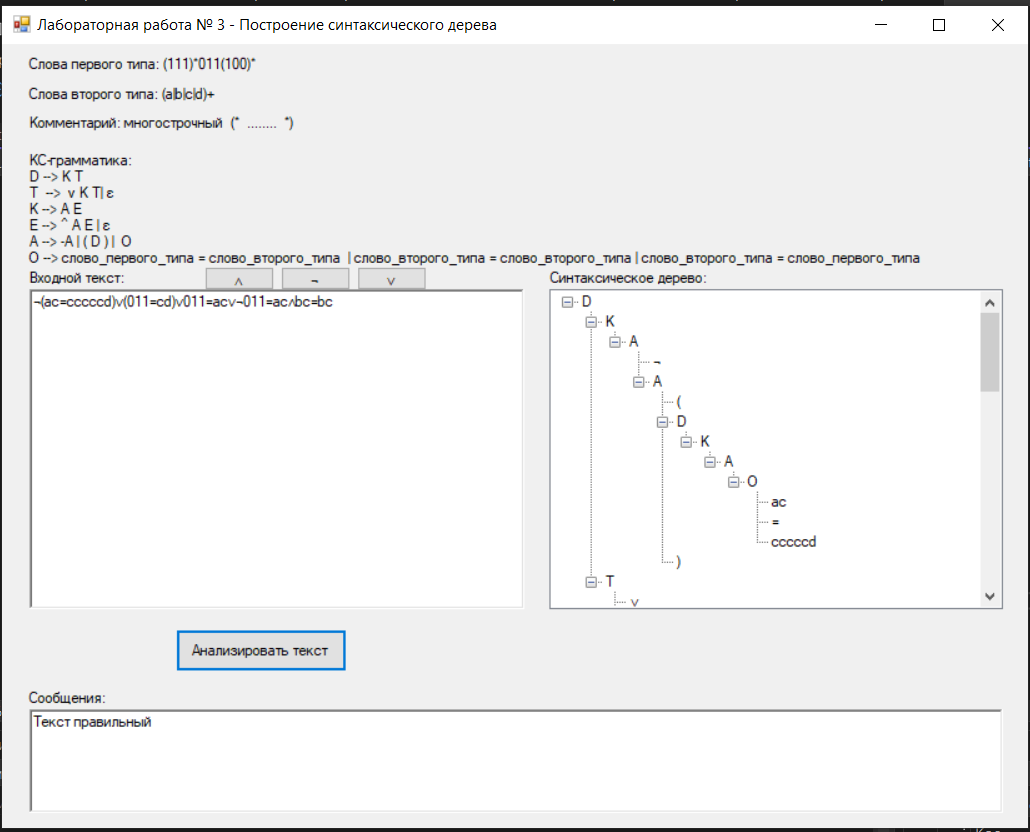


Рисунок 4.1 – Пример работы программы