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 №5

по дисциплине «Формальные языки и трансляторы»

на тему: «Разработка генератора

Выполнил

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

Руководитель

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

г. Ижевск 2022

# Постановка задачи

По результатам анализа исходного текста получить выходной текст в виде: 1. Все числа исходного текста должны быть переведены в десятичное представление. 2. Выполнить вывод полученного текста в структурированном виде с помощью отступов («ступеньками»). Генератор должен осуществлять отдельный просмотр текста (синтаксического дерева). Правила структурирования текста разработать самостоятельно.

# Атрибутная транслирующая грамматика

В таблице 2.1 показана атрибутная транслирующая грамматика.

|  |  |
| --- | --- |
| Продукция | Семантические правила |
| D → K T | D. ↑ val = K. ↑val + T. ↑val |
| T → ∨ K T1 | T ↑val = ∨ + K. ↑val + T1 ↑val |
| K → A E | K ↑val = A. ↑val + E. ↑val |
| E → ∧A E1 | E ↑val = ∧ +A ↑val + E1 ↑val |
| A → ¬ A1 | A ↑val = ¬ + A1 ↑val |
| A → (D) | A ↑val = ( + D. ↑ val + ) |
| A → O | A ↑val = O ↑ val |
| O → <1>=<2> | O. ↑val = <1> = <2> |
| O → <2>=<1> | O ↑ val = <2>=<1> |
| O → <2>=<2> | O ↑ val = <2>=<2> |

# Код программы

using System;

using System.Collections.Generic;

using System.Linq;

using System.Text;

using System.Threading.Tasks;

namespace LAB5

{

// Класс "Генератор".

class Generator

{

SyntaxTreeNode treeRoot; // Корень синтаксического дерева.

List<string> outputText; // Выходной текст - список строк.

// Конструктор генератора.

// В качестве параметра поступает корень синтаксического дерева.

public Generator(SyntaxTreeNode treeRoot)

{

this.treeRoot = treeRoot;

outputText = new List<string>();

}

// Выходной текст - свойство только для чтения.

public List<string> OutputText

{

get { return outputText; }

}

// Сгенерировать структурированный текст.

public void GenerateStructuredText()

{

outputText.Clear(); // Очищаем выходной текст.

RecurTraverseTree(treeRoot, 0); // Рекурсивно обходим дерево и формируем выходной текст.

}

// Рекурсивно обойти дерево, формируя выходной текст.

// node - узел дерева.

// indent - отступ.

private void RecurTraverseTree(SyntaxTreeNode node, int indent)

{

if (node.SubNodes.Count() > 0) // Если текущий узел - нетерминал.

{

foreach(SyntaxTreeNode item in node.SubNodes) // Цикл по всем подчиненным узлам.

{

if (item.Token!=null && item.Token.Type==TokenKind.RightParenthesis) indent -= 10;

RecurTraverseTree(item, indent);

if (item.Token != null && item.Token.Type == TokenKind.LeftParenthesis) indent += 10;

}

}

else

{

var str = new StringBuilder();

var k = 0;

if (node.Token != null &&

(node.Token.Type == TokenKind.Negative

|| node.Token.Type == TokenKind.LogicSum

|| node.Token.Type == TokenKind.LogicMultiply)) k = 5;

// Генерируем отступ.

for(int i = 0; i < indent+k; i++)

{

str.Append(" ");

}

if(node.Token != null && node.Token.Type == TokenKind.FirstWord)

{

var intValue = 0;

for (int i= node.Token.Value.Length-1; i>=0; i--)

{

if(int.Parse(node.Token.Value[i].ToString())==1)

intValue +=(int) Math.Pow(2, node.Token.Value.Length - 1 -i);

}

str.Append(intValue);

outputText.Add(str.ToString());

return;

}

// Добавляем имя узла.

str.Append(node.Name);

// Добавляем созданную строку в результат.

outputText.Add(str.ToString());

}

}

}

}

# Пример работы программы

На рисунке 4.1 показан пример работы программы

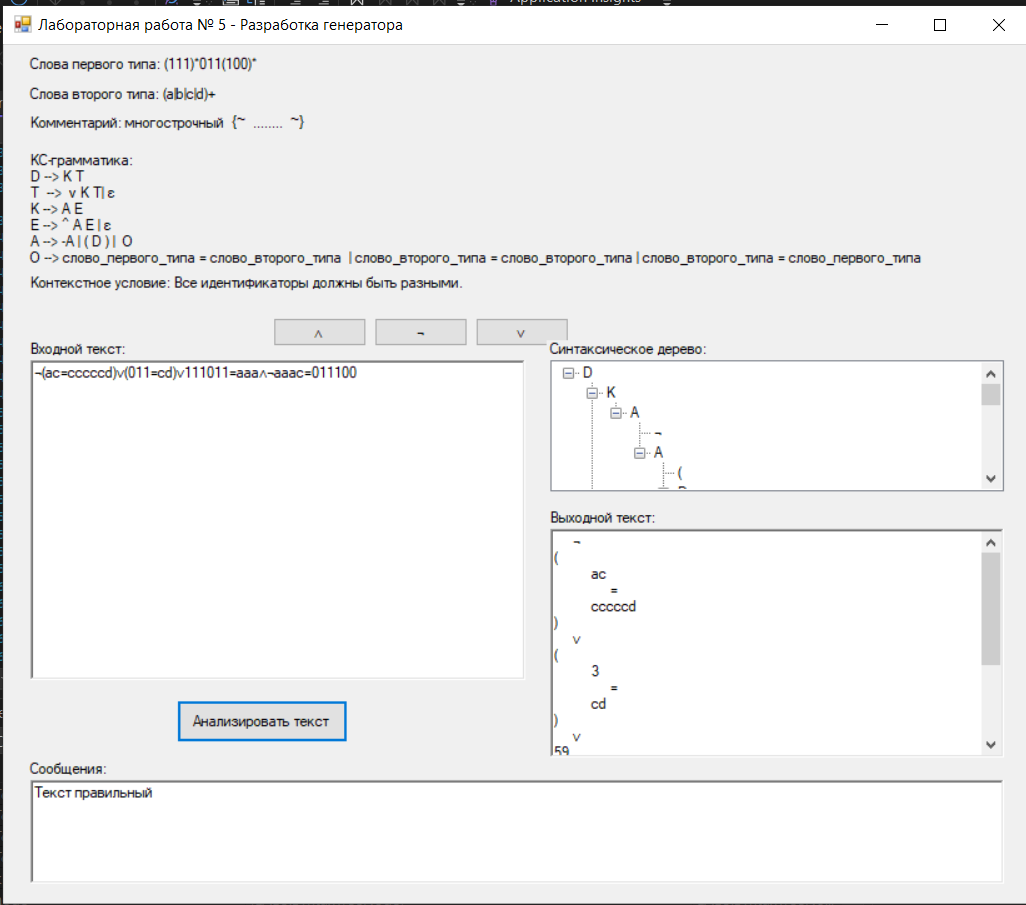


Рисунок 4.1 – Пример работы программы