

**Lab Task # 6**

**Submitted By:**

Ramsha Kokab

FA21-BCS-010

**Course Instructor:**

Mr. Syed Bilal Bukhari

**Course:**

Compiler Construction

**Date:**

11th Octuber, 2024

**DEPARTMENT OF COMPUTER SCIENCE**

**COMSATS UNIVERSITY ISLAMABAD, ATTOCK CAMPUS**

**Lab Task :** Write a code for any given grammar that satisfy the criterion of JAVA language constructs.

.

using System;

using System.Collections;

using System.Collections.Generic;

using System.ComponentModel;

using System.Data;

using System.Drawing;

using System.Linq;

using System.Text;

using System.Text.RegularExpressions;

using System.Threading.Tasks;

using System.Windows.Forms;

namespace Lab6Task\_First\_Sets\_

{

public partial class Form1 : Form

{

public Form1()

{

InitializeComponent();

}

Hashtable productionRules = new Hashtable();

Hashtable firstSets = new Hashtable();

private void button1\_Click(object sender, EventArgs e)

{

productionRules.Clear();

firstSets.Clear();

bool isValid = true; // Track validity of non-terminals

// Read input from the richTextBox1 (grammar rules)

var inputRules = richTextBox1.Text.Split('\n');

foreach (var rule in inputRules)

{

var parts = rule.Split('>'); // Split by '>'

if (parts.Length < 2)

{

MessageBox.Show("Invalid grammar rule format.");

return;

}

string nonTerminal = parts[0].Trim();

string production = parts[1].Trim();

// Validate non-terminal using regex (must start with a letter or underscore)

if (!(new Regex(@"^[A-Za-z\_][A-Za-z0-9\_]\*$")).IsMatch(nonTerminal))

{

isValid = false;

MessageBox.Show("Non-terminals must start with a letter or underscore and contain only alphanumeric characters.");

return;

}

// Add production rules to the hashtable

if (!productionRules.Contains(nonTerminal))

{

productionRules.Add(nonTerminal, production);

}

else

{

productionRules[nonTerminal] += "|" + production;

}

}

if (isValid)

{

// Calculate First sets for all non-terminals

foreach (DictionaryEntry entry in productionRules)

{

string nonTerminal = entry.Key.ToString();

string[] productions = entry.Value.ToString().Split('|');

foreach (var prod in productions)

{

string[] symbols = prod.Split(' ');

if (!firstSets.Contains(nonTerminal))

{

firstSets.Add(nonTerminal, CalculateFirst(symbols, 0));

}

else

{

firstSets[nonTerminal] += "," + CalculateFirst(symbols, 0);

}

}

}

// Display the First sets in richTextBox2

foreach (DictionaryEntry entry in firstSets)

{

richTextBox2.AppendText($"First({entry.Key}) = {{{entry.Value}}}\n");

}

}

}

// Recursive function to calculate the First set

private string CalculateFirst(string[] symbols, int index)

{

if (symbols.Length == 0) return "~"; // Handle epsilon

string symbol = symbols[index];

// If the symbol is a terminal, return it

if (!productionRules.Contains(symbol) && symbol != "~")

{

return symbol;

}

else if (symbol != "~" && symbols.Length > index)

{

string[] production = productionRules[symbol].ToString().Split(' ');

// Recursively calculate the First set

string first = CalculateFirst(production, 0);

if (first != "~")

{

return first;

}

else

{

return CalculateFirst(symbols, index + 1);

}

}

return "~";

}

}

}

**Output:**

