

**LAB # 10**

**Submitted By:**

Alveena

FA21-BCS-014

**Course Instructor:**

Mr. Syed Bilal Haider Bukhari

**Course:**

CC- [Lab]

**Date:**

15th Nov, 2024

**DEPARTMENT OF COMPUTER SCIENCE**

**COMSATS UNIVERSITY ISLAMABAD, ATTOCK CAMPUS**

using System;

using System.Collections.Generic;

namespace SLRParser

{

class Program

{

// Token class to represent individual tokens

public class Token

{

public string Type { get; }

public string Value { get; }

public Token(string type, string value)

{

Type = type;

Value = value;

}

public override string ToString() => $"{Type}: {Value}";

}

// Lexer to tokenize the input string

public static List<Token> Tokenize(string input)

{

var tokens = new List<Token>();

var keywords = new HashSet<string> { "begin", "end", "int", "float", "if", "for", "else", "then", "print" };

var operators = new HashSet<string> { "+", "=", "<" };

var punctuations = new HashSet<char> { '{', '}', '(', ')', ';' };

string[] lines = input.Split(new[] { '\n', ' ' }, StringSplitOptions.RemoveEmptyEntries);

foreach (var word in lines)

{

if (keywords.Contains(word))

{

tokens.Add(new Token("KEYWORD", word));

}

else if (operators.Contains(word))

{

tokens.Add(new Token("OPERATOR", word));

}

else if (punctuations.Contains(word[0]))

{

tokens.Add(new Token("PUNCTUATION", word));

}

else if (char.IsDigit(word[0]) || word.Contains("."))

{

tokens.Add(new Token("CONSTANT", word));

}

else

{

tokens.Add(new Token("IDENTIFIER", word));

}

}

return tokens;

}

// SLR Parsing Table

public static Dictionary<(int, string), string> CreateParsingTable()

{

return new Dictionary<(int, string), string>

{

// Example parsing table entries

{(0, "begin"), "S5"},

{(0, "S"), "1"},

{(1, "$"), "Accept"},

{(2, "int"), "S6"},

{(3, ";"), "S8"},

{(4, "print"), "S12"}

// Add all necessary states and transitions

};

}

// SLR Parsing Simulation

public static void ParseInput(List<Token> tokens, Dictionary<(int, string), string> parsingTable)

{

var stack = new Stack<int>();

stack.Push(0); // Initial state

int index = 0;

Console.WriteLine("Parsing Process:");

Console.WriteLine($"{"Stack",-20} {"Input",-30} {"Action",-10}");

while (true)

{

int state = stack.Peek();

string tokenType = index < tokens.Count ? tokens[index].Type : "$";

if (!parsingTable.ContainsKey((state, tokenType)))

{

Console.WriteLine("Error: Unexpected token");

break;

}

string action = parsingTable[(state, tokenType)];

Console.WriteLine($"{string.Join(" ", stack),-20} {string.Join(" ", tokens.GetRange(index, tokens.Count - index)),-30} {action,-10}");

if (action.StartsWith("S"))

{

// Shift action

stack.Push(int.Parse(action.Substring(1)));

index++;

}

else if (action.StartsWith("R"))

{

// Reduce action (example: R1 -> reduce using rule 1)

int ruleNumber = int.Parse(action.Substring(1));

Console.WriteLine($"Reduce using rule {ruleNumber}");

// Perform stack adjustments and reductions based on the rule

}

else if (action == "Accept")

{

Console.WriteLine("Input successfully parsed!");

break;

}

}

}

static void Main(string[] args)

{

// Input string

string input = @"

begin(){

int a=5;

int b=10;

int c=0;

c=a+b;

if(c>a)

print a;

else print c;

}end";

// Tokenize input

var tokens = Tokenize(input);

Console.WriteLine("Tokens:");

foreach (var token in tokens)

{

Console.WriteLine(token);

}

// Create parsing table

var parsingTable = CreateParsingTable();

// Parse input

ParseInput(tokens, parsingTable);

}

}

}

**Output:**

