**МИНИСТЕРСТВО НАУКИ И ВЫСШЕГО ОБРАЗОВАНИЯ РОССИЙСКОЙ ФЕДЕРАЦИИ**

федеральное государственное бюджетное образовательное учреждение высшего образования «Казанский национальный исследовательский технический университет им. А.Н. Туполева-КАИ»

Кафедра автоматизированных систем обработки и управления

Лабораторная работа №3

по дисциплине «Методы трансляции»

по теме «Разработка контекстно-свободного (КС) синтаксического анализатора»

Вариант 3

|  |  |
| --- | --- |
| Выполнил: | студент группы 4309 Габдуллин С. Ж. |
| Проверил: | Бикмуллина И. И.  Старший преподаватель кафедры АСОИУ |

Казань 2023

**Текст задания:**

Для предложенного преподавателем варианта КС-грамматики разработать методом рекурсивного спуска синтаксический анализатор

**Грамматика: Избавление от левой рекурсии:**

|  |  |  |
| --- | --- | --- |
| S → <2> A<1> B |  | S → <2> A<1> B |
| A → [ A ] |  | A →[ A ] | <1> |
| A → <1> |  | B → <2> | C |
| B → B <2> |  | C →<2> | <2> C |
| B → <2> |  |  |

В строке «B→B <2>» присутствует левая рекурсия

Возможные варианты: S => <2> A<1> B => <2> <1> <1> <2>

S => <2> A<1> B => <2> [<1>] <1> <2> <2>

**uMainForm.cs:**

using System;

using System.Collections.Generic;

using System.ComponentModel;

using System.Data;

using System.Drawing;

using System.Linq;

using System.Text;

using System.Threading.Tasks;

using System.Windows.Forms;

using nsLex;

using nsSynt;

namespace nsLexMainForm

{

public partial class Form1 : Form

{

public Form1()

{

InitializeComponent();

// textBox1.Lines.Count = 2;

textBox1.AppendText("abcd\r\n");

textBox1.AppendText("110 111110 110000 bad\r\n");

textBox1.AppendText(" cb ab");

// int n = textBox1.Lines.Length;

// n = n;

}

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

{

/\*CLex Lex = new CLex();

Lex.strPSource = textBox1.Lines;

Lex.strPMessage = textBox2.Lines;

Lex.enumPState = TState.Start;

textBox2.Text = "";

try

{

while (Lex.enumPState != TState.Finish)

{

Lex.NextToken();

отладка String s = "";

switch (Lex.enumPToken)

{

case TToken.lxmIdentifier: { s = "Identifier"; break; }

case TToken.lxmNumber: { s = "Numder"; break; }

case TToken.lxmEmpty: { s = "EndText"; break; }

}

textBox2.Text += "(" + Lex.strPLexicalUnit + "," + s + ")";

}

textBox2.Text = "Текст верный";

}

catch (Exception exc)

{

textBox2.Text += exc.Message;

textBox1.Select();

textBox1.SelectionStart = 0;

int n = 0;

for (int i = 0; i < Lex.intPSourceRowSelection; i++) n += textBox1.Lines[i].Length + 2;

n += Lex.intPSourceColSelection;

textBox1.SelectionLength = n;

}\*/

textBox2.Clear();

uSyntAnalyzer Synt = new uSyntAnalyzer();

Synt.Lex.strPSource = textBox1.Lines;

Synt.Lex.strPMessage = textBox2.Lines;

Synt.Lex.enumPState = TState.Start;

try {

Synt.S();

}

catch (Exception exc) {

textBox2.Text += exc.Message;

textBox1.Select();

textBox1.SelectionStart = 0;

int n = 0;

for (int i = 0; i < Synt.Lex.intPSourceRowSelection; i++) n += textBox1.Lines[i].Length + 2;

n += Synt.Lex.intPSourceColSelection;

textBox1.SelectionLength = n;

}

}

}

}

**uSyntAnalyzer.cs:**

using System;

using System.Collections.Generic;

using System.Text;

using nsLex;

using nsLexMainForm;

namespace nsSynt

{

class uSyntAnalyzer

{

private String[] strFSource;

private String[] strFMessage;

public String[] strPSource { set { strFSource = value; } get { return strFSource; } }

public String[] strPMessage { set { strFMessage = value; } get { return strFMessage; } }

public CLex Lex = new CLex();

public void S()

{

if (Lex.enumPToken == TToken.lxmUpperIdentifier)

{

Lex.NextToken();

}

else throw new Exception("Ожидалось идентификатор");

A();

if (Lex.enumPToken == TToken.lxmNumber)

{

Lex.NextToken();

}

else throw new Exception("Ожидалось число");

B();

throw new Exception("Конец слова, текст верный");

}

public void A()

{

Lex.NextToken();

if (Lex.enumPToken == TToken.lxmls)

{

A();

if (Lex.enumPToken == TToken.lxmrs)

{

Lex.NextToken();

}

else throw new Exception("Ожидалась ]");

}

else if (Lex.enumPToken == TToken.lxmNumber)

{

Lex.NextToken();

}

else throw new Exception("Ожидалось [ или идентификатор");

}

public void C()

{

if (Lex.enumPToken == TToken.lxmUpperIdentifier)

{

Lex.NextToken();

C();

throw new Exception("Конец слова, текст верный.");

}

else throw new Exception("Ожидалась идентификатор");

}

public void B()

{

Lex.NextToken();

if (Lex.enumPToken == TToken.lxmUpperIdentifier)

{

Lex.NextToken();

}

else

{

Lex.NextToken();

C();

}

}

}

}

**uLex.cs:**

using System;

using nsLexMainForm;

namespace nsLex

{

public enum TState { Start, Continue, Finish };

public enum TCharType { Letter, Digit, EndRow, EndText, Space, ReservedSymbol };

public enum TToken { lxmUpperIdentifier, lxmNumber, lxmUnknown, lxmUl, lxmEmpty, lxmLeftParenth, lxmRightParenth, lxmIs, lxmDot, lxmComma, lxmText, lxmdt, lxmr, lxmrs, lxmls};

public class CLex

{

private String[] strFSource; // указатель на массив строк

private String[] strFMessage; // указатель на массив строк

public TCharType enumFSelectionCharType;

private char chrFSelection;

private TState enumFState;

private int intFSourceRowSelection;

private int intFSourceColSelection;

private String strFLexicalUnit;

private TToken enumFToken;

public String[] strPSource { set { strFSource = value; } get { return strFSource; } }

public String[] strPMessage { set { strFMessage = value; } get { return strFMessage; } }

public TState enumPState { set { enumFState = value; } get { return enumFState; } }

public String strPLexicalUnit { set { strFLexicalUnit = value; } get { return strFLexicalUnit; } }

public TToken enumPToken { set { enumFToken = value; } get { return enumFToken; } }

public int intPSourceRowSelection { get { return intFSourceRowSelection; } }

public int intPSourceColSelection { get { return intFSourceColSelection; } }

public CLex()

{

}

public void GetSymbol(bool comm)

{

intFSourceColSelection++;

if (intFSourceColSelection > strFSource[intFSourceRowSelection].Length - 1)

{

intFSourceRowSelection++;

if (intFSourceRowSelection <= strFSource.Length - 1)

{

intFSourceColSelection = -1;

chrFSelection = '\0';

enumFSelectionCharType = TCharType.EndRow;

enumFState = TState.Continue;

}

else

{

chrFSelection = '\0';

enumFSelectionCharType = TCharType.EndText;

enumFState = TState.Finish;

}

}

else

{

chrFSelection = strFSource[intFSourceRowSelection][intFSourceColSelection];

if (chrFSelection == ' ') enumFSelectionCharType = TCharType.Space;

else if (chrFSelection >= 'a' && chrFSelection <= 'd') enumFSelectionCharType = TCharType.Letter;

else if (chrFSelection == '0' || chrFSelection == '1') enumFSelectionCharType = TCharType.Digit;

else if (chrFSelection == '[' || chrFSelection == ']' || chrFSelection == ':' || chrFSelection == '-' || chrFSelection == ',' || chrFSelection == '.' || chrFSelection == '\_')

enumFSelectionCharType = TCharType.ReservedSymbol;

else if (!comm) throw new System.Exception("Cимвол вне алфавита");

enumFState = TState.Continue;

}

}

private void TakeSymbol()

{

char[] c = { chrFSelection };

String s = new string(c);

strFLexicalUnit += s;

GetSymbol(false);

}

public void NextToken()

{

strFLexicalUnit = "";

if (enumFState == TState.Start)

{

intFSourceRowSelection = 0;

intFSourceColSelection = -1;

GetSymbol(false);

}

while (enumFSelectionCharType == TCharType.Space || enumFSelectionCharType == TCharType.EndRow)

{

GetSymbol(false);

}

if (chrFSelection == '/')

{

GetSymbol(false);

if (chrFSelection == '/')

while (enumFSelectionCharType != TCharType.EndRow)

{

GetSymbol(true);

}

GetSymbol(false);

}

// Вариант 3

switch (enumFSelectionCharType)

{

case TCharType.Letter:

{

// a b c d

// A |BFin|CFin|BFin|BFin|

// BFin |BFin|CFin|BFin|BFin|

// CFin |BFin| |BFin|BFin|

A:

{

if (chrFSelection == 'a' || chrFSelection == 'c' || chrFSelection == 'd')

{

TakeSymbol();

goto BFin;

}

else if (chrFSelection == 'b')//|| chrFSelection == 'b' || chrFSelection == 'd')

{

TakeSymbol();

goto CFin;

}

else throw new Exception("Ошибка в идентификаторе");

}

BFin:

{

if (chrFSelection == 'a' || chrFSelection == 'c' || chrFSelection == 'd')

{

TakeSymbol();

goto BFin;

}

else if (chrFSelection == 'b')

{

TakeSymbol();

goto CFin;

}

else

{

enumFToken = TToken.lxmUpperIdentifier;

return;

}

}

CFin:

{

if (chrFSelection == 'a' || chrFSelection == 'c' || chrFSelection == 'd')

{

TakeSymbol();

goto BFin;

}

else if (chrFSelection == 'b')

{

throw new Exception("Не соблюдено условие грамматики: Не должно быть подстроки bb");

}

else

{

enumFToken = TToken.lxmUpperIdentifier;

return;

}

}

}

case TCharType.Digit:

{

// 0 1

// A | B | |

// B | C | D |

// C | A | |

// D |EFin| |

// EFin | F | |

// F | | G |

// G | |EFin|

A:

if (chrFSelection == '0')

{

TakeSymbol();

goto B;

}

else throw new Exception("Ожидалось 1");

B:

if (chrFSelection == '0')

{

TakeSymbol();

goto C;

}

else if (chrFSelection == '1')

{

TakeSymbol();

goto D;

}

else throw new Exception("Ожидалось продолжение");

C:

if (chrFSelection == '0')

{

TakeSymbol();

goto A;

}

// else if (chrFSelection == '0')

// {

// TakeSymbol();

// goto DFin;

//}

else throw new Exception("Ожидалась цифра 1");

D:

if (chrFSelection == '0')

{

TakeSymbol();

goto EFin;

}

// else if (enumFSelectionCharType != TCharType.Digit) { enumFToken = TToken.lxmNumber; return; }

else throw new Exception("Ожидалось 0");

EFin:

if (chrFSelection == '0')

{

TakeSymbol();

goto F;

}

else

if (chrFSelection == '1') throw new Exception("Ожидалось 0 или конец слова");

else { enumFToken = TToken.lxmNumber; return; }

//if (enumFSelectionCharType != TCharType.Digit) { enumFToken = TToken.lxmNumber; return; }

//else throw new Exception("Ожидалось 1");

F:

if (chrFSelection == '1')

{

TakeSymbol();

goto G;

}

else throw new Exception("Ожидалось 1");

G:

if (chrFSelection == '1')

{

TakeSymbol();

goto EFin;

}

else throw new Exception("Ожидалось 1");

}

case TCharType.ReservedSymbol:

{

if (chrFSelection == '/')

{

GetSymbol(false);

if (chrFSelection == '/')

while (enumFSelectionCharType != TCharType.EndRow)

{

GetSymbol(true);

}

GetSymbol(false);

}

if (chrFSelection == '(')

{

enumFToken = TToken.lxmLeftParenth;

GetSymbol(false);

return;

}

if (chrFSelection == ')')

{

enumFToken = TToken.lxmRightParenth;

GetSymbol(false);

return;

}

if (chrFSelection == '[')

{

enumFToken = TToken.lxmls;

GetSymbol(false);

return;

}

if (chrFSelection == ']')

{

enumFToken = TToken.lxmrs;

GetSymbol(false);

return;

}

if (chrFSelection == ',')

{

enumFToken = TToken.lxmComma;

GetSymbol(false);

return;

}

if (chrFSelection == ':')

{

enumFToken = TToken.lxmdt;

GetSymbol(false);

return;

}

if (chrFSelection == '=')

{

enumFToken = TToken.lxmr;

GetSymbol(false);

return;

}

if (chrFSelection == '\_')

{

enumFToken = TToken.lxmUl;

GetSymbol(false);

return;

}

break;

}

case TCharType.EndText:

{

enumFToken = TToken.lxmEmpty;

break;

}

case TCharType.Space:

{

GetSymbol(false);

break;

}

}

}

}

} Результаты выполнения программы:





