Министерство образования Российской Федерации

Федеральное агентство по образованию

КАЗАНСКИЙ НАЦИОНАЛЬНЫЙ ИССЛЕДОВАТЕЛЬСКИЙ ТЕХНИЧЕСКИЙ УНИВЕРСИТЕТ

им. А.Н. Туполева - КАИ

Кафедра АСОИУ

Лабораторная работа №3

по дисциплине

«ТЕОРИЯ ЯЗЫКОВ ПРОГРАММИРОВАНИЯ»

Выполнила:

Студентка группы 4309

Хафизов Н.Р.

Проверила: Бикмуллина И.И.

Казань, 2023

**Текст задания:**

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

**Грамматика:**

O → A = A | (L)

A → <1> | <2>

L→ True | False | U | not L | O

U → <2>

**Ulex.cs**

using System;

using System.Collections.Generic;

using System.Linq;

using System.Text;

using System.Threading.Tasks;

namespace nsLex

{

public enum TState { Start, Continue, Finish }; //тип состояния

public enum TCharType { Letter, Digit, EndRow, EndText, Space, ReservedSymbol, LBracket, RBracket, Exclamation, Semicolon, Coma, Question, Not, True, False }; // тип символа

public enum TToken { lxmIdentifier, lxmNumber, lxmUnknown, lxmEmpty, lxmLeftParenth, lxmRightParenth, lxmIs, lxmDot, lxmComma, lxmText, lxmtz, lxmdt, lxmr, lxmrs, lxmls, lxmTrue, lxmFalse, lxmNot};

public class CLex

{

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

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

public TCharType enumFSelectionCharType;

public 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; } set { intFSourceRowSelection = value; } }

public int intPSourceColSelection

{

get { return intFSourceColSelection; }

set { intFSourceColSelection = value; }

}

public CLex()

{

}

public void GetSymbol() //метод класса лексический анализатор

{

intFSourceColSelection++; // продвигаем номер колонки

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]; //классификация прочитанной литеры

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 == '/') enumFSelectionCharType = TCharType.ReservedSymbol;

else if (chrFSelection == '\*') enumFSelectionCharType = TCharType.ReservedSymbol;

else if (chrFSelection == '!') enumFSelectionCharType = TCharType.Exclamation;

else if (chrFSelection == ';') enumFSelectionCharType = TCharType.Semicolon;

else if (chrFSelection == ',') enumFSelectionCharType = TCharType.Coma;

else if (chrFSelection == '?') enumFSelectionCharType = TCharType.Question;

else if (chrFSelection == ';' || chrFSelection == ',' || chrFSelection == '[' || chrFSelection == ']' || chrFSelection == '=' || chrFSelection == ':' || chrFSelection == '(' || chrFSelection == ')' || chrFSelection == 'T' || chrFSelection == 'F' || chrFSelection == 'N') enumFSelectionCharType = TCharType.ReservedSymbol;

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

enumFState = TState.Continue;

}

}

private void TakeSymbol()

{

char[] c = { chrFSelection };

String s = new string(c);

strFLexicalUnit += s;

GetSymbol();

}

public void NextToken()

{

strFLexicalUnit = "";

if (enumFState == TState.Start)

{

intFSourceRowSelection = 0;

intFSourceColSelection = -1;

GetSymbol();

}

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

{

GetSymbol();

}

if (chrFSelection == '/')

{

GetSymbol();

if (chrFSelection == '/')

while (enumFSelectionCharType != TCharType.EndRow)

{

GetSymbol();

}

GetSymbol();

}

// Вариант 24

switch (enumFSelectionCharType)

{

case TCharType.Letter:

{

// a b c d

// A | B | B | B | B |

// B | | C | | |

// C |DFin| | | |

// DFin |DFin|DFin|DFin|DFin|

A:

{

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

{

TakeSymbol();

goto B;

}

}

B:

{

if (chrFSelection == 'b')

{

TakeSymbol();

goto C;

}

else throw new Exception("Вторые два символа всегда ba");

}

C:

{

if (chrFSelection == 'a')

{

TakeSymbol();

goto DFin;

}

else throw new Exception("Вторые два символа всегда ba");

}

DFin:

{

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

{

TakeSymbol();

goto DFin;

}

else

{

enumFToken = TToken.lxmIdentifier;

return;

}

}

}

if (chrFSelection == '/')

{

GetSymbol();

if (chrFSelection == '/')

while (enumFSelectionCharType != TCharType.EndRow)

{

GetSymbol();

}

GetSymbol();

}

case TCharType.Digit:

{

// 0 1

// A | B | C |

// B | | D |

// C | E | |

// D | | A |

// E | |FFin |

// FFin | | G |

// G | | H |

// H |FFin | |

A:

if (chrFSelection == '0')

{

TakeSymbol();

goto B;

}

else if (chrFSelection == '1')

{

TakeSymbol();

goto C;

}

else throw new Exception("Ожидался 0 или 1");

B:

if (chrFSelection == '1')

{

TakeSymbol();

goto D;

}

else throw new Exception("Ожидался 1");

C:

if (chrFSelection == '0')

{

TakeSymbol();

goto E;

}

else throw new Exception("Ожидался 0");

D:

if (chrFSelection == '1')

{

TakeSymbol();

goto A;

}

else throw new Exception("Ожидался 1");

E:

if (chrFSelection == '1')

{

TakeSymbol();

goto FFin;

}

else throw new Exception("Ожидалась 1");

FFin:

if (chrFSelection == '1')

{

TakeSymbol();

goto G;

}

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

else throw new Exception("Ожидалась 1");

G:

if (chrFSelection == '1')

{

TakeSymbol();

goto H;

}

else throw new Exception("Ожидался 1");

H:

if (chrFSelection == '0')

{

TakeSymbol();

goto FFin;

}

else throw new Exception("Ожидался 0");

}

case TCharType.ReservedSymbol:

{

if (chrFSelection == '/')

{

GetSymbol();

if (chrFSelection == '/')

{

while (enumFSelectionCharType != TCharType.EndRow)

GetSymbol();

}

GetSymbol();

}

if (chrFSelection == '(')

{

enumFToken = TToken.lxmLeftParenth;

GetSymbol();

return;

}

if (chrFSelection == ')')

{

enumFToken = TToken.lxmRightParenth;

GetSymbol();

return;

}

if (chrFSelection == '[')

{

enumFToken = TToken.lxmls;

GetSymbol();

return;

}

if (chrFSelection == ']')

{

enumFToken = TToken.lxmrs;

GetSymbol();

return;

}

if (chrFSelection == ',')

{

enumFToken = TToken.lxmComma;

GetSymbol();

return;

}

if (chrFSelection == ':')

{

enumFToken = TToken.lxmdt;

GetSymbol();

return;

}

if (chrFSelection == '=')

{

enumFToken = TToken.lxmr;

GetSymbol();

return;

}

if (chrFSelection == 'T')

{

enumFToken = TToken.lxmTrue;

GetSymbol();

return;

}

if (chrFSelection == 'F')

{

enumFToken = TToken.lxmFalse;

GetSymbol();

return;

}

if (chrFSelection == 'N')

{

enumFToken = TToken.lxmNot;

GetSymbol();

return;

}

break;

}

case TCharType.EndText:

{

enumFToken = TToken.lxmEmpty;

break;

}

}

}

}

}

**uSyntAnalyzer.cs**

using System;

using System.Collections.Generic;

using System.Linq;

using System.Text;

using System.Threading.Tasks;

using nsLex;

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 O()

{

if (Lex.enumPToken == TToken.lxmNumber || Lex.enumPToken == TToken.lxmIdentifier)

{

A();

if (Lex.enumPToken == TToken.lxmr)

{

Lex.NextToken();

A();

}

else throw new Exception("Ожидал =");

}

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

{

Lex.NextToken();

L();

if (Lex.enumPToken == TToken.lxmRightParenth)

{

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

}

else throw new Exception("Ожидалась )");

}

else throw new Exception("Ожидалась (");

}

public void A()

{

if (Lex.enumPToken == TToken.lxmNumber || Lex.enumPToken == TToken.lxmIdentifier)

{

Lex.NextToken();

}

else throw new Exception("ozhidal word or numb");

}

public void L()

{

if (Lex.enumPToken == TToken.lxmTrue || Lex.enumPToken == TToken.lxmFalse)

{

Lex.NextToken();

}

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

{

U();

}

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

{

Lex.NextToken();

L();

}

else

{

O();

}

}

public void U()

{

if (Lex.enumPToken == TToken.lxmIdentifier)

{

Lex.NextToken();

}

else throw new Exception("Ожидалось слово");

}

}

}





