Лабораторная работа №1

Выполнил: студент группы 4309

Курбанов Д.И.

Лексический анализатор – программа, принимающая на вход текст (последовательность символов из алфавита) и разбивающая его на подстроки (лексемы) в соответствии с некоторым набором регулярных выражений.

Текст задания:

Спроектировать и отладить экранную форму для ввода исходных данных, вывода сообщений программы и управления программой.

Разработать и отладить транслитератор void GetSymbol(), пример имеется в модуле uLexicalAnalizer из папки «Программы».

Для отладки транслитератора временно включить в обработчик нажатия кнопки цикл чтения с помощью функции GetSymbol() символов исходного текста и вывода результатов анализа в поле диагностических сообщений.

Код программы:

Class1.cs

using System;

using System.Collections.Generic;

using System.Linq;

using System.Text;

using System.Threading.Tasks;

namespace WindowsFormsApp2

{

public enum TState { Start, Continue, Finish };

public enum TCharType { Letter, Digit, EndRow, EndText, Space, ReservedSymbol, Undefined, Opendoor,Closedoor, point, talepoint,talepointPoint };

public enum TToken { lxmIdentifier, lxmNumber, lxmUnknown, lxmEmpty, lxmLeftParenth, lxmRightParenth, lxmIs, lxmDot, lxmComma };

public class CLex

{

private String[] strFSource;

private String[] strFMessage;

public TCharType enumFSelectionCharType;

public char chrFSelection;

private TState enumFState;

private int intFSourceRowSelection;

private int intFSourceColSelection = -1;

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()

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

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

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

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

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

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

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

else if (chrFSelection == '!' || chrFSelection == '+' || chrFSelection == ';' || chrFSelection == '$' || chrFSelection == '!' || chrFSelection == ':' || chrFSelection == '-' || chrFSelection == ',' ) enumFSelectionCharType = TCharType.ReservedSymbol;

else

{

enumFSelectionCharType = TCharType.Undefined;

}

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();

}

if (chrFSelection == '/')

{

GetSymbol();

if (chrFSelection == '/')

while (enumFSelectionCharType != TCharType.EndRow)

{

GetSymbol();

}

GetSymbol();

}

}

}

}

Form.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;

namespace WindowsFormsApp2

{

public partial class Form1 : Form

{

public Form1()

{

InitializeComponent();

tbFSource.AppendText("");

int n = tbFSource.Lines.Length;

}

private void Form1\_Load(object sender, EventArgs e)

{

}

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

{

CLex Lex = new CLex();

Lex.strPSource = tbFSource.Lines;

Lex.strPMessage = tbFMessage.Lines;

Lex.intPSourceColSelection = -1;

Lex.intPSourceRowSelection = 0;

int x = tbFSource.TextLength;

int y = tbFSource.Lines.Length;

tbFMessage.Text = "";

try

{

Lex.GetSymbol();

while (Lex.enumPState != TState.Finish)

{

Lex.NextToken();

String s = "";

String s1 = "";

switch (Lex.enumFSelectionCharType)

{

case TCharType.Letter: { s1 = "Letter"; break; }

case TCharType.Digit: { s1 = "Digit"; break; }

case TCharType.Space: { s1 = "Space"; break; }

case TCharType.ReservedSymbol: { s1 = "ReservedSymbol"; break; }

case TCharType.EndRow: { s = "KC"; s1 = "EndRow"; break; }

case TCharType.EndText: { s = "KT"; s1 = "EndText"; break; }

case TCharType.Undefined: { s1 = "Undefined"; break; }

case TCharType.point: { s1 = "point"; break; }

case TCharType.Opendoor: { s1 = "Opendoor"; break; }

case TCharType.Closedoor: { s1 = "Closedoor"; break; }

case TCharType.talepointPoint: { s1 = "talepointPoint"; break; }

case TCharType.talepoint: { s1 = "talepoint"; break; }

}

switch (Lex.enumPToken)

{

case TToken.lxmNumber: { s = "Номер"; s1 = Lex.strPLexicalUnit; break; }

case TToken.lxmIdentifier: { s = "Слово"; s1 = Lex.strPLexicalUnit; break; }

}

String m = "(" + s + "," + s1 + ")";

tbFMessage.Text += m;

}

}

catch (Exception exc)

{

tbFMessage.Text += exc.Message;

tbFSource.Select();

tbFSource.SelectionStart = 0;

int n = 0;

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

n += Lex.intPSourceColSelection;

tbFSource.SelectionLength = n;

}

}

private void tbFMessage\_TextChanged(object sender, EventArgs e)

{

}

private void tbFSource\_TextChanged(object sender, EventArgs e)

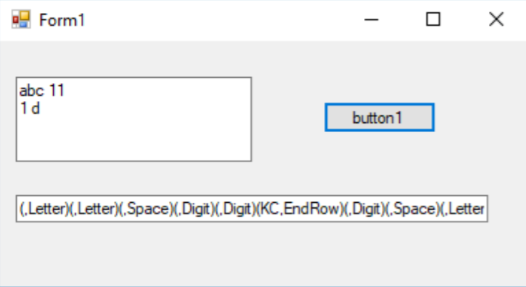
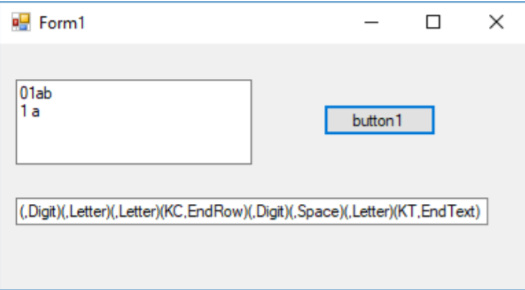
{

}

}

}

Результат работы программы:



**Лабораторная работа №2**

Выполнил: студент группы 4309

Курбанов Д.И.

Лексический анализатор предназначен для чтения слов в исходном тексте и классификации прочитанных слов.

Основные функции лексического анализатора:

Чтение с помощью транслитератора очередного слова в исходном

тексте и его класси-фикация;

Пропуск пробелов и комментариев;

Выдача диагностических сообщений об обнаруженных

лексических ошибках.

Токен – некоторый класс слов (выделяется из потребностей синтаксического анализа в основном по принципу построения и назначения слов).

Лексический анализатор – программа, принимающая на вход текст (последовательность символов из алфавита) и разбивающая его на подстроки (лексемы) в соответствии с некоторым набором регулярных выражений. Последовательность лексем такова, что каждая лексема принадлежит хотя бы одному из языков, задаваемых регулярными выражениями.

В детерминированном алгоритме для данного конкретного входного сигнала компьютер всегда будет выдавать один и тот же выходной сигнал, проходящий через одни и те же состояния, но в случае недетерминированного алгоритма для одного и того же входного сигнала компилятор может выдавать разные выходные данные в разных запусках. На самом деле, недетерминированные алгоритмы не могут решить проблему за полиномиальное время и не могут определить, каков следующий шаг. Недетерминированные алгоритмы могут демонстрировать разное поведение для одних и тех же входных данных при разном выполнении, и в этом есть определенная степень случайности.

Элементарные операции над матрицами - это операции, выполняемые над строками и столбцами матрицы, которые не изменяют значение матрицы. Матрица - это способ представления чисел в виде массива, т.е. числа расположены в виде строк и столбцов. В матрице строки и столбцы содержат все значения в ячейке. Мы представляем матрицу в виде [A]m × n, где A - матрица, m - количество строк в матрице, n - количество столбцов матрицы.

**Текст задания:**

1. Спроектировать и отладить экранную форму для ввода исходных данных, вывода сообщений программы и управления программой.
2. Включить из лабораторной работы № 1 транслитератор **void GetSymbol().**
3. Составить регулярную грамматику для каждого вида слов.
4. Построить конечные автоматы для каждого вида слов, как правило, они будут недетерминированными.
5. Построить детерминированные конечные автоматы для каждого вида слов.
6. Составить объединенный конечный автомат.
7. Написать и отладить модуль лексического анализатора по алгоритму объединенного конечного автомата. Для чтения исходного текста использовать транслитератор. Предусмотреть обработчик лексических ошибок исходного текста, используется конструкция **try … catch**.

Для отладки лексического анализатора временно включить в обработчик нажатия кнопки цикл чтения слов исходного текста и вывода результатов лексического анализа.

**Первое слово:**

(010)\*000(001)\*

A-> 0B | 0С

B-> 1D

С-> 0E

D-> 0A

E-> 0F | 0

F-> 0G

G-> 0H

H -> 1 | 1F

Детерминированная :

|  |  |  |
| --- | --- | --- |
|  | 0 | 1 |
| A | BC |  |
| BC | E | D |
| D | A |  |
| E | FFin |  |
| FFin | G |  |
| G | H |  |
| H |  | FFin |

Недетерминированная :

|  |  |  |
| --- | --- | --- |
|  | 0 | 1 |
| A | BC |  |
| B |  | D |
| C | E |  |
| D | A |  |
| E | FFin |  |
| F | G |  |
| G | H |  |
| H |  | FFin |
| Fin |  |  |

Второе слово:

(a | b | c | d)\*

A-> cB

B-> aC

C-> bC | aC | cC | dC

Недетерминированная:

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
|  | a | b | c | d |
| A |  |  | B |  |
| B | C,Fin |  |  |  |
| C | C,Fin | C,Fin | C,Fin | C,Fin |
| Fin |  |  |  |  |

Детерминированная:

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
|  | a | b | c | d |
| A |  |  | B |  |
| B | CFin |  |  |  |
| CFin | CFin | CFin | CFin | CFin |

текст программы:

using System;

using System.Collections.Generic;

using System.Text;

namespace \_2\_ая\_лаба

{

public enum TState { Start, Continue, Finish };

public enum TCharType { Letter, Digit, EndRow, EndText, Space, ReservedSymbol, Undefined, Opendoor, Closedoor, point, talepoint, talepointPoint };

public enum TToken { lxmIdentifier, lxmNumber, lxmUnknown, lxmEmpty, lxmLeftParenth, lxmRightParenth, lxmIs, lxmDot, lxmComma, lxmText };

public class CLex

{

private String[] strFSource;

private String[] strFMessage;

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 void GetSymbol()

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

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

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

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

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

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

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

else if (chrFSelection == '!' || chrFSelection == '+' || chrFSelection == ';' || chrFSelection == '$' || chrFSelection == '!' || chrFSelection == ':' || chrFSelection == '-' || chrFSelection == ',') 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 14

switch (enumFSelectionCharType)

{

case TCharType.Letter:

{

// a b c d

// A | | | B | |

// B |CFin| | | |

// CFin |CFin|CFin|CFin|CFin|

A:

{

if (chrFSelection == 'c')

{

TakeSymbol();

goto B;

}

else throw new Exception("Слово должно начинаться с 'ca'");

}

B:

{

if (chrFSelection == 'a')

{

TakeSymbol();

goto CFin;

}

else throw new Exception("Слово должно начинаться с 'ca'");

}

CFin:

{

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

{

TakeSymbol();

goto CFin;

}

else

{

enumFToken = TToken.lxmIdentifier;

return;

}

}

}

if (chrFSelection == '/')

{

GetSymbol();

if (chrFSelection == '/')

while (enumFSelectionCharType != TCharType.EndRow)

{

GetSymbol();

}

GetSymbol();

}

case TCharType.Digit:

{

// 0 1

// A | BC | |

// BC | E | D |

// D | A | |

// E |FFin | |

// FFin | G | |

// G | | G |

A:

if (chrFSelection == '0')

{

TakeSymbol();

goto B;

}

else throw new Exception("Ожидался 0");

B:

if (chrFSelection == '0')

{

TakeSymbol();

goto E;

}

else if (chrFSelection == '1')

{

TakeSymbol();

goto D;

}

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

D:

if (chrFSelection == '0')

{

TakeSymbol();

goto A;

}

else throw new Exception("Ожидался 0");

E:

if (chrFSelection == '0')

{

TakeSymbol();

goto FFin;

}

else throw new Exception("Ожидалось 0");

FFin:

if (chrFSelection == '0')

{

TakeSymbol();

goto G;

}

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

else throw new Exception("Ожидалось 0");

G:

if (chrFSelection == '0')

{

TakeSymbol();

goto H;

}

else throw new Exception("Ожидался 0");

H:

if (chrFSelection == '1')

{

TakeSymbol();

goto FFin;

}

else throw new Exception("Ожидался 1");

}

case TCharType.ReservedSymbol:

{

if (chrFSelection == '/')

{

GetSymbol();

if (chrFSelection == '/')

{

while (enumFSelectionCharType != TCharType.EndRow)

GetSymbol();

}

GetSymbol();

}

break;

}

case TCharType.EndText:

{

enumFToken = TToken.lxmEmpty;

break;

}

}

}

}

}

Результат:



Лабораторная работа №3

Выполнил: студент группы 4309

Курбанов Д.И.

Синтаксический анализатор – это программное обеспечение, которое анализирует структуру текста и определяет его соответствие грамматике, что позволяет автоматически проверять и разбирать программный код, упрощая процесс разработки и отладки.

Синтаксический анализатор, также известный как парсер, является одной из ключевых компонент программного обеспечения, используемого в компиляторах и интерпретаторах. Он выполняет анализ входного текста, проверяет его на соответствие определенной грамматике и создает структуру данных, называемую синтаксическим деревом или абстрактным синтаксическим деревом (AST).

Синтаксический анализатор играет важную роль в процессе компиляции или интерпретации программного кода. Он принимает входной текст, который может быть написан на определенном языке программирования, и проверяет его на наличие синтаксических ошибок. Если текст соответствует грамматике языка, синтаксический анализатор создает структуру данных, которая представляет собой иерархическое представление программы.

Основная цель синтаксического анализатора – разбор входного текста и создание структуры данных, которая будет использоваться в дальнейшем для выполнения семантического анализа и генерации исполняемого кода. Он помогает программистам и разработчикам понять структуру программы и обнаружить возможные ошибки в синтаксисе.

Левая рекурсия устраняется путем преобразования грамматики в праворекурсивную грамматику.

Если имеются правила грамматики следующего вида

A -> A α | β,

то это леворекурсивная грамматика. Цепочка β не начинается с нетерминала A.

Левая рекурсия устраняется с помощью введения дополнительного нетерминала А':

А -> β А'

А' -> α А'

| ϵ,

где ϵ – пустая цепочка. Это праворекурсивная грамматика, которая порождает тот же язык,

что и исходная грамматика.

**Вариант 14**

**Текст задания**:

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

**Грамматика:**

S−−> <1> A S

S−−> <2>

A−−> <1>

A−−> <1>

Левая рекурсия отсутствует

S−−> **<1> <2> <2> <1> <2>**

S−−> **<1> <1> <2>**

Код программы:

UsyntAnalyzer

using System;

using System.Collections.Generic;

using System.Linq;

using System.Text;

using System.Threading.Tasks;

namespace WindowsFormsApp4

{

class Class1

{

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();

TToken first;

bool init;

public void S()

{

Lex.NextToken();

if (!init && (Lex.enumPToken == TToken.lxmIdentifier || Lex.enumPToken == TToken.lxmNumber))

{

first = Lex.enumPToken;

init = true;

}

if (Lex.enumPToken == first)

{

A();

S();

}

else if (Lex.enumPToken == (TToken)(1 - (int)first))

{

}

else throw new Exception("Ожидался число или идентификатор");

}

public void A()

{

Lex.NextToken();

if (Lex.enumPToken == first)

{

}

else if (Lex.enumPToken == (TToken)(1 - (int)first))

{

S();

A();

}

else throw new Exception("Ожидался буквенный идентификатор");

}

}

}

ULex

using System;

using System.Collections.Generic;

using System.Linq;

using System.Text;

using System.Threading.Tasks;

namespace WindowsFormsApp4

{

public enum TState { Start, Continue, Finish };

public enum TCharType { Letter, Digit, EndRow, EndText, Space, ReservedSymbol, Undefined, Opendoor, Closedoor, point, talepoint, talepointPoint };

public enum TToken { lxmIdentifier, lxmNumber, lxmUnknown, lxmEmpty, lxmLeftParenth, lxmRightParenth, lxmIs, lxmDot, lxmComma, lxmText };

public class CLex

{

private String[] strFSource;

private String[] strFMessage;

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 void GetSymbol()

{

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

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

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

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

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

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

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

else if (chrFSelection == '!' || chrFSelection == '+' || chrFSelection == ';' || chrFSelection == '$' || chrFSelection == '!' || chrFSelection == ':' || chrFSelection == '-' || chrFSelection == ',') 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 14

switch (enumFSelectionCharType)

{

case TCharType.Letter:

{

// a b c d

// A | | | B | |

// B |CFin| | | |

// CFin |CFin|CFin|CFin|CFin|

A:

{

if (chrFSelection == 'c')

{

TakeSymbol();

goto B;

}

else throw new Exception("Слово должно начинаться с 'ca'");

}

B:

{

if (chrFSelection == 'a')

{

TakeSymbol();

goto CFin;

}

else throw new Exception("Слово должно начинаться с 'ca'");

}

CFin:

{

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

{

TakeSymbol();

goto CFin;

}

else

{

enumFToken = TToken.lxmIdentifier;

return;

}

}

}

if (chrFSelection == '/')

{

GetSymbol();

if (chrFSelection == '/')

while (enumFSelectionCharType != TCharType.EndRow)

{

GetSymbol();

}

GetSymbol();

}

case TCharType.Digit:

{

// 0 1

// A | BC | |

// BC | E | D |

// D | A | |

// E |FFin | |

// FFin | G | |

// G | | G |

A:

if (chrFSelection == '0')

{

TakeSymbol();

goto B;

}

else throw new Exception("Ожидался 0");

B:

if (chrFSelection == '0')

{

TakeSymbol();

goto E;

}

else if (chrFSelection == '1')

{

TakeSymbol();

goto D;

}

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

D:

if (chrFSelection == '0')

{

TakeSymbol();

goto A;

}

else throw new Exception("Ожидался 0");

E:

if (chrFSelection == '0')

{

TakeSymbol();

goto FFin;

}

else throw new Exception("Ожидалось 0");

FFin:

if (chrFSelection == '0')

{

TakeSymbol();

goto G;

}

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

else throw new Exception("Ожидалось 0");

G:

if (chrFSelection == '0')

{

TakeSymbol();

goto H;

}

else throw new Exception("Ожидался 0");

H:

if (chrFSelection == '1')

{

TakeSymbol();

goto FFin;

}

else throw new Exception("Ожидался 1");

}

case TCharType.ReservedSymbol:

{

if (chrFSelection == '/')

{

GetSymbol();

if (chrFSelection == '/')

{

while (enumFSelectionCharType != TCharType.EndRow)

GetSymbol();

}

GetSymbol();

}

break;

}

case TCharType.EndText:

{

enumFToken = TToken.lxmEmpty;

break;

}

}

}

}

}

Form1:

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;

namespace WindowsFormsApp4

{

public partial class Form1 : Form

{

public Form1()

{

InitializeComponent();

tbFSource.AppendText("010000001 010000001 caaaa");

int n = tbFSource.Lines.Length;

}

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

{

{

Class1 Synt = new Class1();

Synt.Lex.strPSource = tbFSource.Lines;

Synt.Lex.strPMessage = tbFMessage.Lines;

Synt.Lex.intPSourceColSelection = -1;

Synt.Lex.intPSourceRowSelection = 0;

int x = tbFSource.TextLength;

int y = tbFSource.Lines.Length;

tbFMessage.Text = "";

try

{

tbFMessage.Clear();

Synt.S();

throw new Exception("Конец слова.");

}

catch (Exception exc)

{

tbFMessage.Text += exc.Message;

tbFSource.Select();

tbFSource.SelectionStart = 0;

int n = 0;

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

n += Synt.Lex.intPSourceColSelection;

tbFSource.SelectionLength = n;

}

}

}

private void Form1\_Load(object sender, EventArgs e)

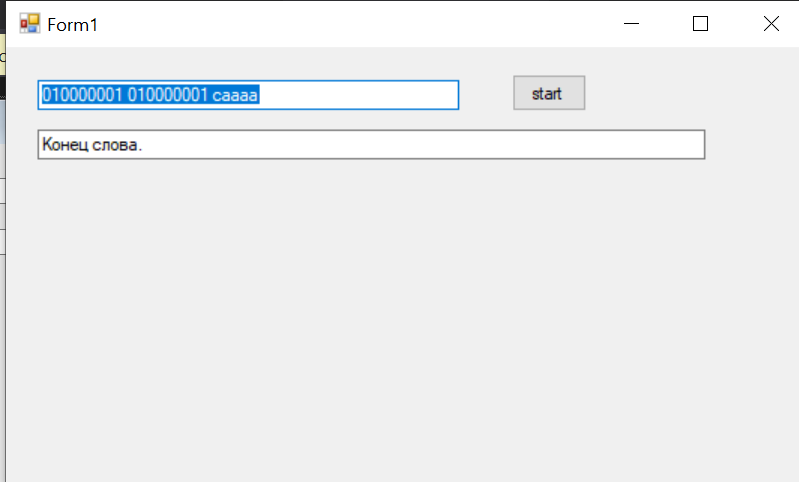
{

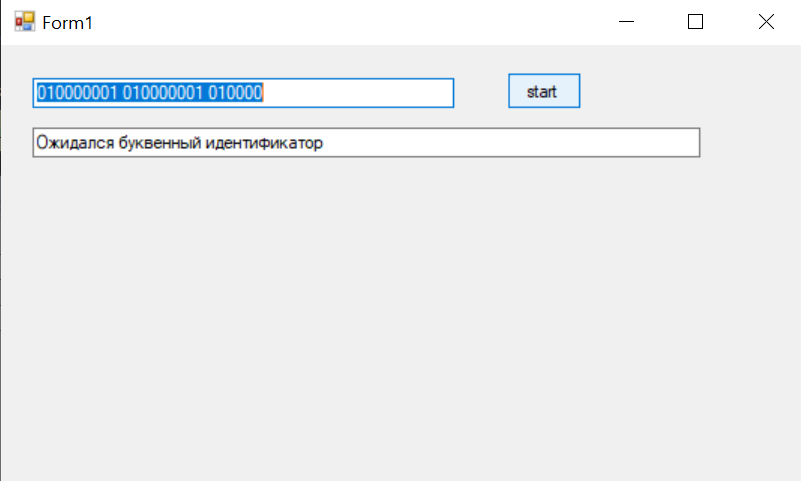
}

}

}

Результат:





Лабораторная работа № 4

Выполнил: студент группы 4309

Курбанов Д.И.

Основное назначение хеширования — проверка информации. Эта задача важна в огромном количестве случаев: от проверки паролей на сайте до сложных вычислений в блокчейне. Так как хеш — это уникальный код определенного набора данных, по нему можно понять, соответствует ли информация ожидаемой. Поэтому программа может хранить хеши вместо образца данных для сравнения. Это может быть нужно для защиты чувствительных сведений или экономии места.

Хеш-табли́ца — это структура данных, реализующая интерфейс ассоциативного массива, а именно, она позволяет хранить пары (ключ, значение) и выполнять три операции: операцию добавления новой пары, операцию поиска и операцию удаления пары по ключу. Существуют два основных варианта хеш-таблиц: с цепочками и открытой адресацией.

Хеш-функция— функция, осуществляющая преобразование массива входных данных произвольной длины в выходную битовую строку установленной длины, выполняемое определённым алгоритмом. Преобразование, производимое хеш-функцией, называется хешированием. Исходные данные называются входным массивом, «ключом» или «сообщением».

Под коллизией понимается ситуация, когда при добавлении разных объектов мы попадаем в одну и ту же ячейку массива. Для разрешения коллизий придумано 2 метода: метод цепочек и метод открытой адресации.

Коллизии существуют для большинства хеш-функций, но для «хороших» хеш-функций частота их возникновения близка к теоретическому минимуму. В некоторых частных случаях, когда множество различных входных данных [конечно](https://ru.wikipedia.org/wiki/%D0%9A%D0%BE%D0%BD%D0%B5%D1%87%D0%BD%D0%BE%D0%B5_%D0%BC%D0%BD%D0%BE%D0%B6%D0%B5%D1%81%D1%82%D0%B2%D0%BE), можно задать [инъективную](https://ru.wikipedia.org/wiki/%D0%98%D0%BD%D1%8A%D0%B5%D0%BA%D1%82%D0%B8%D0%B2%D0%BD%D0%BE%D1%81%D1%82%D1%8C) хеш-функцию, по определению не имеющую коллизий. Однако для хеш-функций, принимающих вход переменной длины и возвращающих хеш постоянной длины, коллизии обязаны существовать, поскольку хотя бы для одного значения хеш-функции соответствующее ему множество входных данных будет бесконечно — и любые два набора данных из этого множества образуют коллизию.

Метод цепочек является наиболее простым методом разрешения коллизий. В ячейке массива мы будем хранить не элементы, а связанный список данных элементов. Потому как добавление в начало списка обладает асимптотикой , мы не испортим общую асимптотику, и она останется равной.

У данной реализации есть проблема: если списки будут очень сильно вырастать (в качестве крайнего случая можно рассмотреть хеш-функцию, которая возвращает константу для любого объекта), то мы получим асимптотику O(m), где m — число элементов во множестве, если размер массива фиксирован. Для избежания таких неприятностей вводится понятие коэффициент заполнения (он может быть равен, например, 1.5). Если при добавлении элемента оказывается, что доля числа элементов, находящихся в структуре данных по отношению к размеру массива, превосходит коэффициент заполнения, то происходит следующее: выделяется новый массив, размер которого превосходит размер старого массива (например в 2 раза), и структура данных перестраивается на новом массиве.

Метод открытой адресации. В данном методе в ячейках хранятся сами элементы, а в случае коллизии происходит последовательность проб, то есть мы начинаем по некоторому алгоритму перебирать ячейки в надежде найти свободную. Это можно делать разными алгоритмами (линейная / квадратичная последовательности проб, двойное хеширование), каждый из которых обладает своими проблемами (например, возникновение первичных или вторичных кластеров).

**Задание**. Синтаксический анализатор размещает распознанные с помощью лексического анализатора слова в таблицы. Количество таблиц соответствует числу типов слов. В нашем случае нужны три таблицы: таблица чисел (первый тип слова), таблица идентификаторов (второй тип слова), таблица составленных из специальных символов слов (третий тип слова). Таблицы организуются методом хеширования. Включить класс «Поиск слова в таблице методом хеширования» в разрабатываемый транслятор.

Код программы:

MyHashFunction.сs:

using System;

using System.Collections.Generic;

using System.Linq;

using System.Text;

using System.Threading.Tasks;

namespace WindowsFormsApp4

{

public class MyHashFunction

{

public int HashFunction(string word)

{

int hashValue = 0;

foreach (char c in word)

{

hashValue += (int)c;

}

return hashValue;

}

public void AddWord(Dictionary<int, List<string>> hashTable, string word)

{

int hashValue = HashFunction(word);

if (!hashTable.ContainsKey(hashValue))

{

hashTable[hashValue] = new List<string>();

}

else

{

if (hashTable[hashValue].Contains(word))

{

Console.WriteLine("Слово уже существует: " + word);

return;

}

}

hashTable[hashValue].Add(word);

}

public int SearchWord(Dictionary<int, List<string>> hashTable, string word)

{

int hashValue = HashFunction(word);

if (hashTable.ContainsKey(hashValue))

{

return 1;

}

else

{

return 0;

}

}

public bool RemoveWord(Dictionary<int, List<string>> hashTable, string word)

{

int hashValue = HashFunction(word);

if (hashTable.ContainsKey(hashValue))

{

List<string> words = hashTable[hashValue];

if (words.Contains(word))

{

words.Remove(word);

if (words.Count == 0)

{

hashTable.Remove(hashValue);

}

return true;

}

}

return false;

}

}

}

Form1.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;

namespace WindowsFormsApp4

{

public partial class Form1 : Form

{

Dictionary<int, List<string>> hashTableIdentifier = new Dictionary<int, List<string>>();

Dictionary<int, List<string>> hashTableDigital = new Dictionary<int, List<string>>();

Dictionary<int, List<string>> hashTableRezerv = new Dictionary<int, List<string>>();

public MyHashFunction hashFunction = new MyHashFunction();

public Form1()

{

InitializeComponent();

//textBox2.AppendText("010000001 010000001 caaaa");

textBox2.AppendText("010000001 caaa cab 010000001 caaaa");

int n = textBox2.Lines.Length;

}

private void Form1\_Load(object sender, EventArgs e)

{

}

private void Form1\_Load\_1(object sender, EventArgs e)

{

}

private void label5\_Click(object sender, EventArgs e)

{

}

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

{

textBox4.Clear();

uSyntAnalyzer Synt = new uSyntAnalyzer();

Synt.Lex.strPSource = textBox2.Lines;

Synt.Lex.strPMessage = textBox4.Lines;

Synt.Lex.enumPState = TState.Start;

Synt.Lex.hashTableRezerv = hashTableRezerv;

try

{

Synt.S();

Synt.Lex.NextToken();

throw new Exception("Текст верный");

}

catch (Exception exc)

{

textBox4.Text += exc.Message;

textBox2.Select();

textBox2.SelectionStart = 0;

int n = 0;

for (int i = 0; i < Synt.Lex.intPSourceRowSelection; i++) n += textBox2.Lines[i].Length + 2;

n += Synt.Lex.intPSourceColSelection;

textBox2.SelectionLength = n;

}

}

private void button2\_Click(object sender, EventArgs e)

{

CLex Lex = new CLex();

Lex.strPSource = textBox2.Lines;

Lex.strPMessage = textBox4.Lines;

Lex.intPSourceColSelection = 0;

Lex.intPSourceRowSelection = 0;

int x = textBox2.TextLength;

int y = textBox2.Lines.Length;

textBox4.Text = "";

try

{

while (Lex.enumPState != TState.Finish)

{

Lex.NextToken();

string s1 = "", s = "";

switch (Lex.enumPToken)

{

case TToken.lxmIdentifier:

{

hashFunction.AddWord(hashTableIdentifier, Lex.strPLexicalUnit);

listBox4.Items.Add(Lex.strPLexicalUnit);

s1 = "id " + Lex.strPLexicalUnit;

break;

}

case TToken.lxmNumber:

{

hashFunction.AddWord(hashTableDigital, Lex.strPLexicalUnit);

listBox5.Items.Add(Lex.strPLexicalUnit);

s1 = "num " + Lex.strPLexicalUnit;

break;

}

case (TToken.lxmtz):

{

hashFunction.AddWord(hashTableRezerv, "\_");

listBox6.Items.Add(Lex.strPLexicalUnit);

s1 = "rez \_" + Lex.strPLexicalUnit;

break;

}

}

String m = "(" + s + "" + s1 + ")";

textBox4.Text += m;

}

}

catch (Exception exc)

{

textBox4.Text += exc.Message;

textBox2.Select();

textBox2.SelectionStart = 0;

int n = 0;

for (int i = 0; i < Lex.intPSourceRowSelection; i++) n += textBox2.Lines[i].Length + 2;

n += Lex.intPSourceColSelection;

textBox2.SelectionLength = n;

}

}

private void button3\_Click(object sender, EventArgs e)

{

if (hashFunction.SearchWord(hashTableIdentifier, listBox4.SelectedItem.ToString()) == 1)

{

button3.BackColor = Color.Green;

}

else

{

button3.BackColor = Color.Red;

}

}

private void button4\_Click(object sender, EventArgs e)

{

hashFunction.AddWord(hashTableIdentifier, textBox3.Text.ToString());

}

private void button5\_Click(object sender, EventArgs e)

{

if (hashFunction.RemoveWord(hashTableIdentifier, listBox4.SelectedItem.ToString()))

{

button5.BackColor = Color.Green;

}

else

{

button5.BackColor = Color.Red;

}

}

private void button6\_Click(object sender, EventArgs e)

{

if (hashFunction.RemoveWord(hashTableIdentifier, listBox4.SelectedItem.ToString()))

{

hashFunction.AddWord(hashTableIdentifier, textBox3.Text.ToString());

button6.BackColor = Color.Green;

}

else

{

button6.BackColor = Color.Red;

}

}

private void button7\_Click(object sender, EventArgs e)

{

listBox4.Items.Clear();

listBox5.Items.Clear();

listBox6.Items.Clear();

foreach (var entry in hashTableIdentifier)

{

listBox4.Items.Add(string.Join(", ", entry.Value));

}

foreach (var entry in hashTableDigital)

{

listBox5.Items.Add(string.Join(", ", entry.Value));

}

foreach (var entry in hashTableRezerv)

{

listBox6.Items.Add(string.Join(", ", entry.Value));

}

}

private void label7\_Click(object sender, EventArgs e)

{

}

private void listBox6\_SelectedIndexChanged(object sender, EventArgs e)

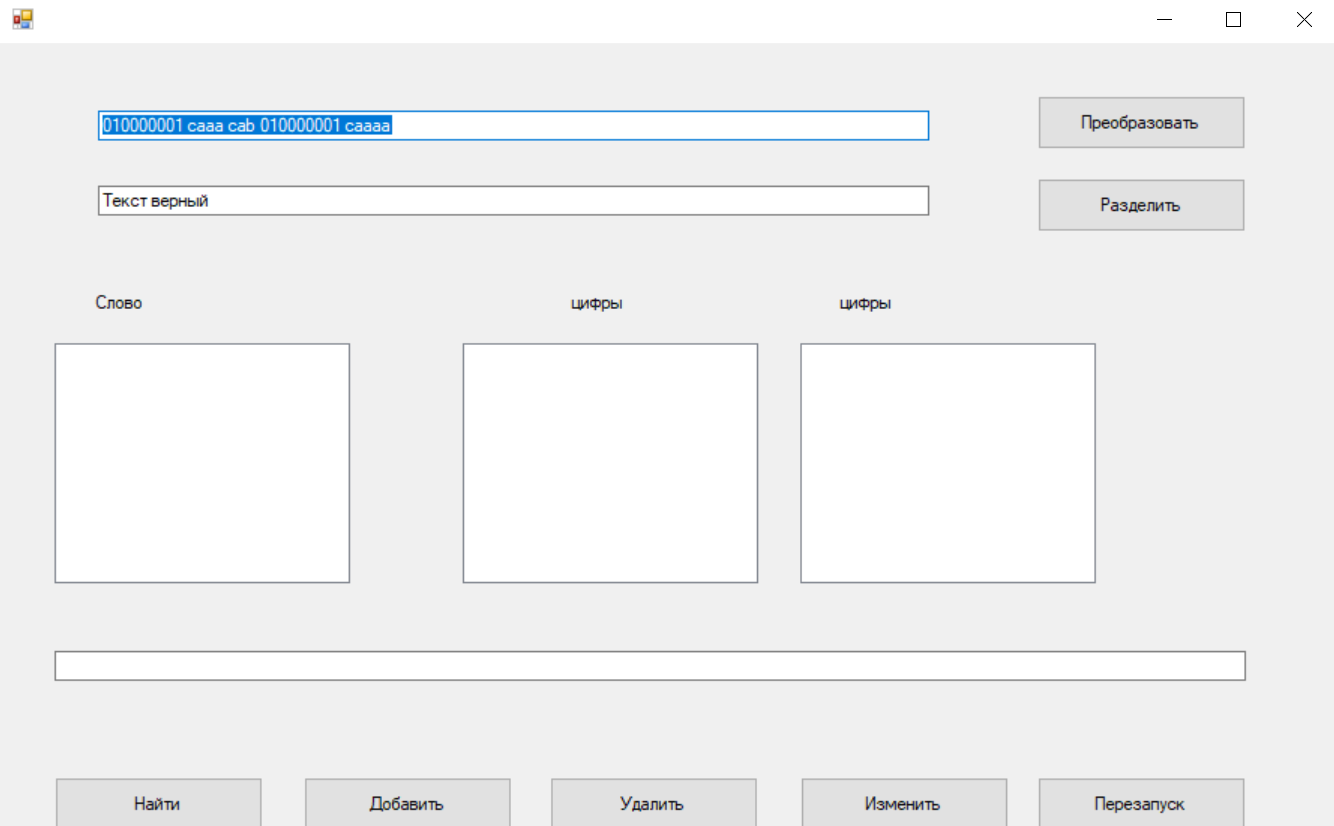
{

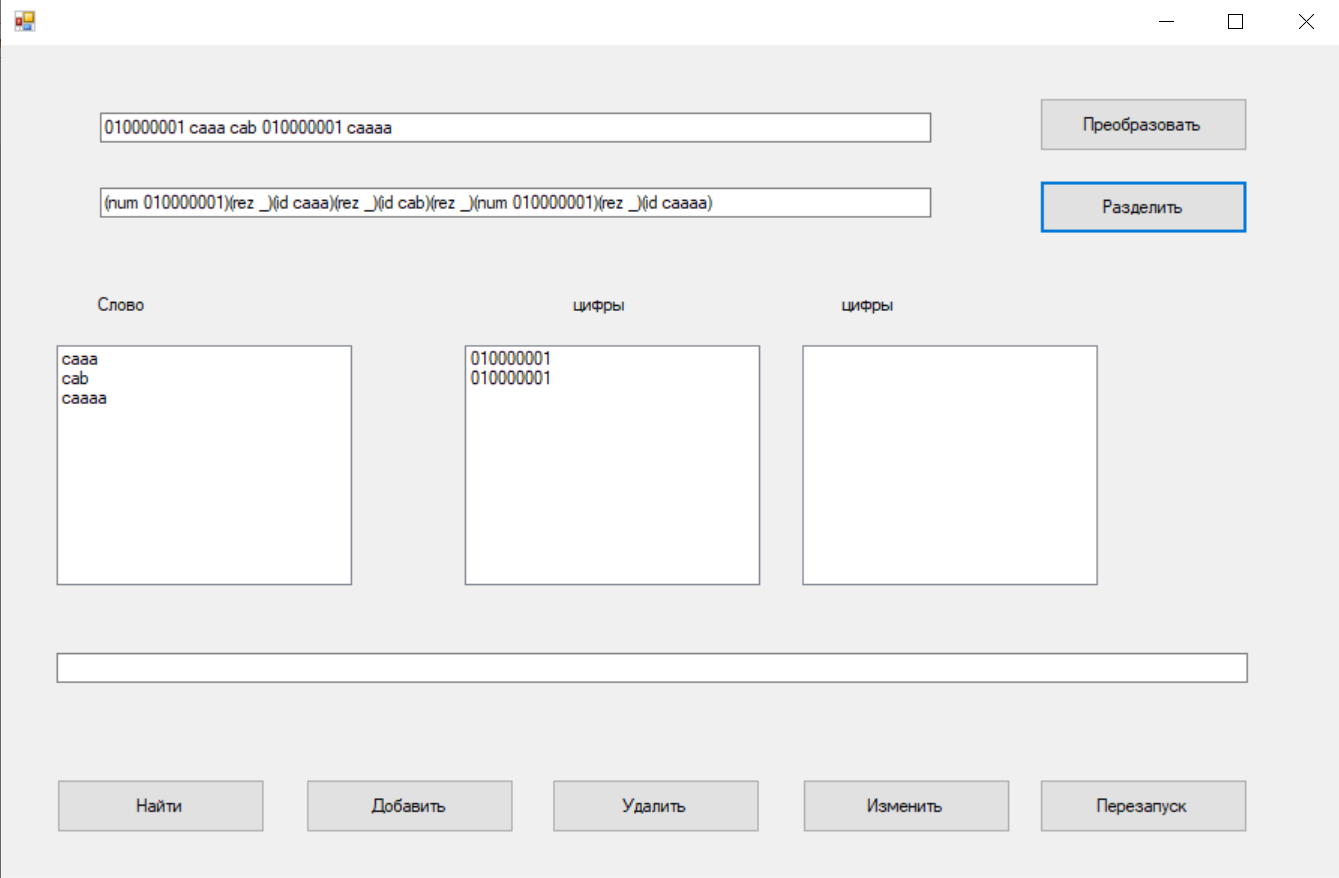
}

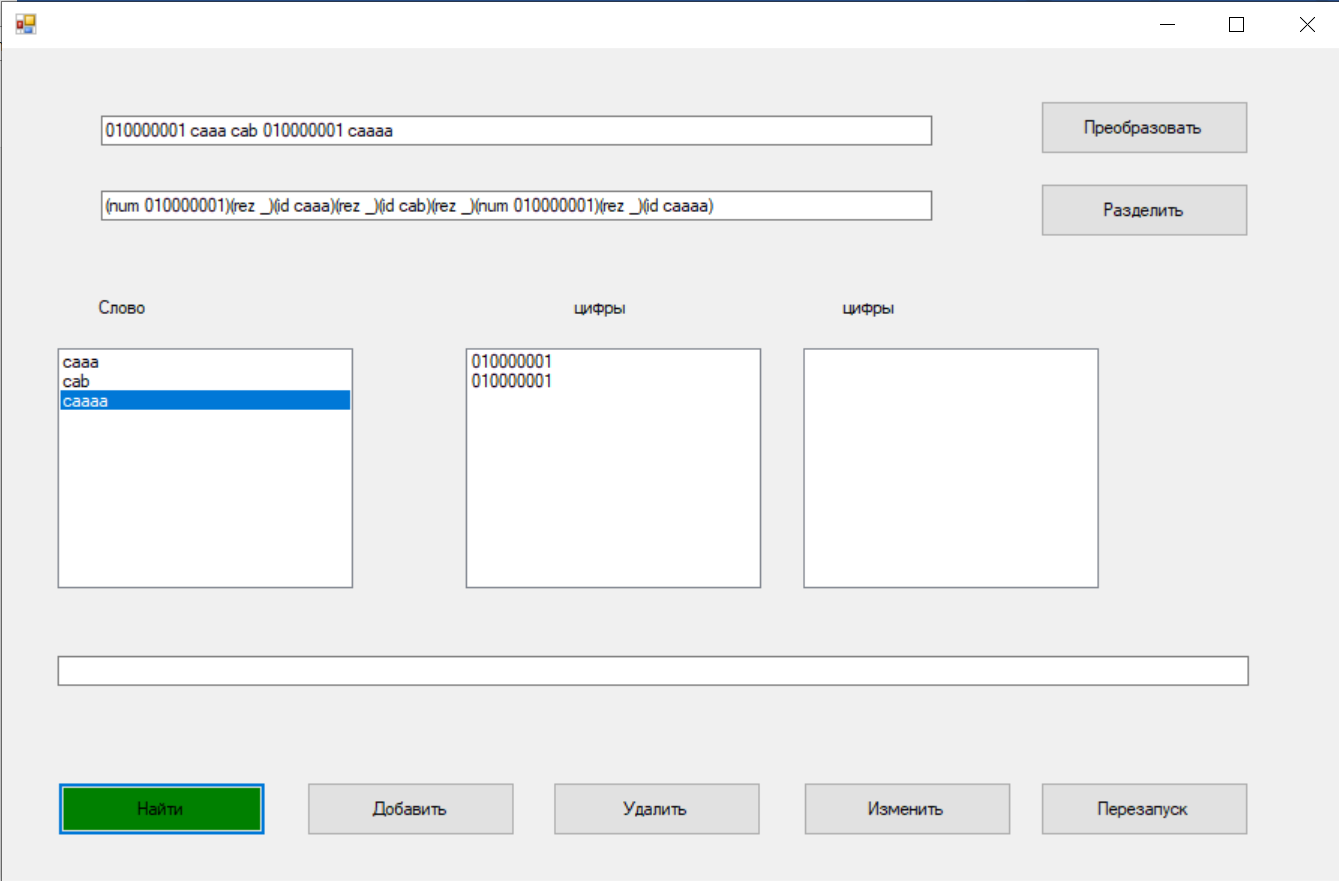
}

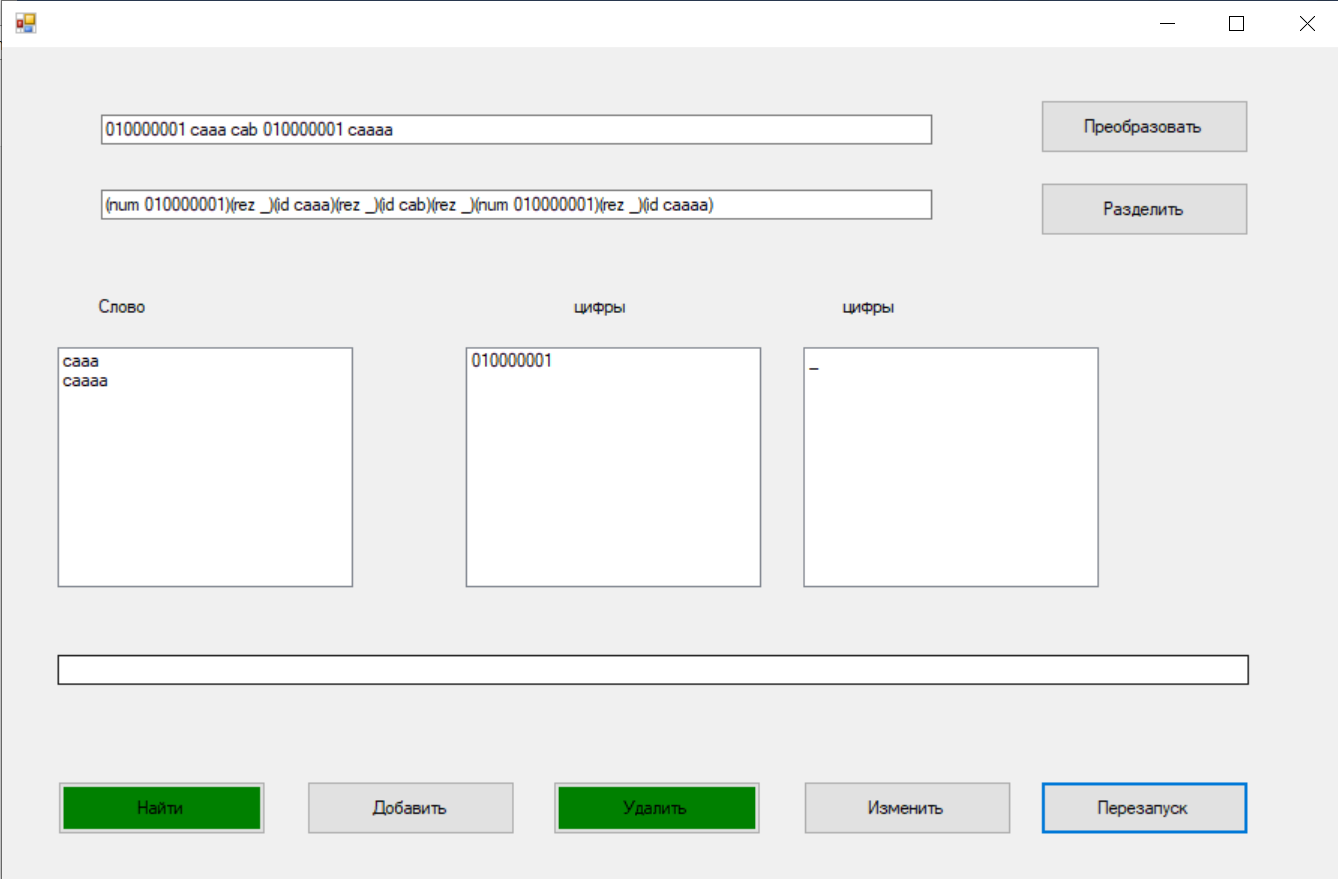
}

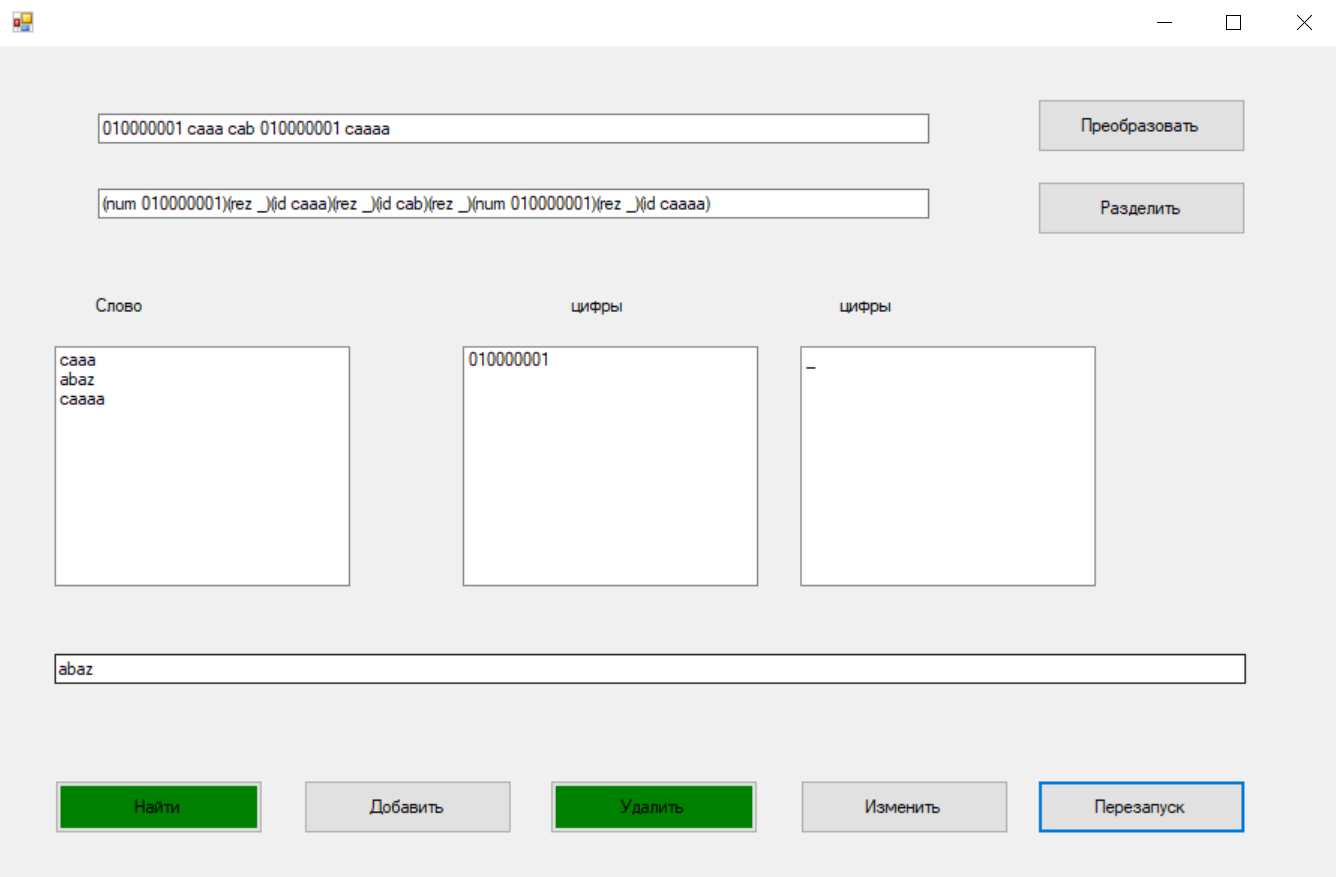
Результаты программы:

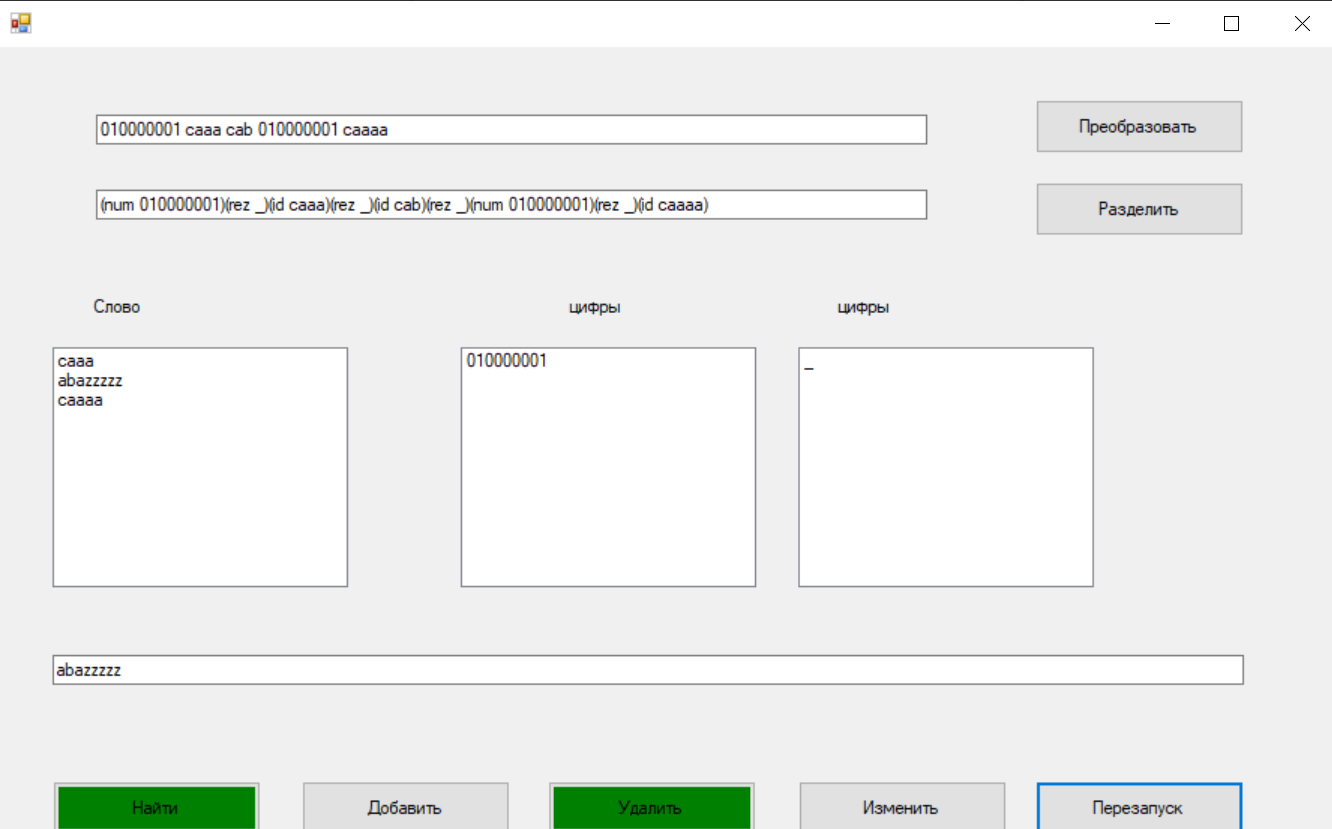












**Лабораторная работа №5**

Выполнил: студент группы 4309

Курбанов Д.И.

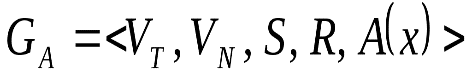
*Атрибутная грамматика* – это КС-грамматика, с узлами дерева вывода которой связаны атрибуты (семантические правила). КС-правилам сопоставляются правила вычисления атрибутов. Правило вычисления значений атрибутов, соответствующее данному КС-правилу, применяется для всех вхождений этого правила в дерево вывода.

Атрибуты могут быть двух видов - *синтезированные и унаследованные*.

*Синтезированные атрибуты* вычисляются с учетом значений атрибутов узлов потомков.

*Унаследованные атрибуты* - это атрибуты, значение которых вычисляется с учетом значений атрибутов его предков.

Формально **атрибутная грамматика** – это пятёрка объектов:

, где

- множество терминальных символов;

- множество нетерминальных символов;

- начальный выделенный символ;

- это правила вывода.

TreeView представляет визуальный элемент в виде дерева. Дерево содержит узлы, которые представляют объекты TreeNode. Узлы могут содержать другие подузлы и могут находиться как скрытом, так и в раскрытом состоянии. Все узлы содержатся в свойстве Nodes.

Внутреннее свойство Nodes, доступное в элементе TreeView возвращает коллекцию узлов дерева, и по умолчанию она пуста. Как видите, чтобы добавить новый элемент, достаточно использовать метод Add(string text) для данного свойства.

Метод Add(string text) возвращает добавленный узел в виде объекта класса TreeNode и к нему точно так же можно обратиться для добавления его дочерних узлов - через свойство Nodes и тот же метод Add(string text).

Синтаксическое дерево - это древовидное представление синтаксической структуры фрагмента исходного кода. Обычно он используется в процессе разработки компилятора, чтобы представить структуру кода таким образом, чтобы его было легче анализировать и манипулировать.

**Задание**. Включить в синтаксический анализатор из лабораторной работы №.3 построение синтаксического дерева. Использовать атрибутный метод Кнута, т.е. преобразовать КС – грамматику из лабораторной работы № 3 в атрибутную грамматику добавлением атрибутов и правил построения синтаксического дерева. Расширить программу синтаксического анализатора из лабораторной работы № 3 введением действий по построению синтаксического дерева.

Код программы:

Analyzer.cs

using System;

using System.Collections.Generic;

using System.Linq;

using System.Text;

using System.Threading.Tasks;

using System.Windows.Forms;

namespace WindowsFormsApp4

{

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();

TToken first;

bool init;

TreeView tree;

public uSyntAnalyzer(TreeView treeView)

{

tree = treeView;

}

public void S(TreeNode highParent)

{

TreeNode parent = new TreeNode("S");

highParent.Nodes.Add(parent);

Lex.NextToken();

if (Lex.enumPToken == TToken.lxmNumber)

{

parent.Nodes.Add(new TreeNode(Lex.strPLexicalUnit));

Lex.NextToken();

A(parent);

S(parent);

}

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

{

parent.Nodes.Add(new TreeNode(Lex.strPLexicalUnit));

Lex.NextToken();

}

else throw new Exception("Ожидался число или идентификатор");

}

public void A(TreeNode highParent)

{

TreeNode parent = new TreeNode("A");

highParent.Nodes.Add(parent);

Lex.NextToken();

if (Lex.enumPToken == TToken.lxmIdentifier)

{

parent.Nodes.Add(new TreeNode(Lex.strPLexicalUnit));

Lex.NextToken();

S(parent);

A(parent);

}

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

{

parent.Nodes.Add(new TreeNode(Lex.strPLexicalUnit));

Lex.NextToken();

}

else throw new Exception("Ожидался число или идентификатор");

}

}

}

Form1.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;

namespace WindowsFormsApp4

{

public partial class Form1 : Form

{

Dictionary<int, List<string>> hashTableIdentifier = new Dictionary<int, List<string>>();

Dictionary<int, List<string>> hashTableDigital = new Dictionary<int, List<string>>();

Dictionary<int, List<string>> hashTableRezerv = new Dictionary<int, List<string>>();

public MyHashFunction hashFunction = new MyHashFunction();

public Form1()

{

InitializeComponent();

//textBox2.AppendText("010000001 010000001 caaaa");

textBox2.AppendText("010000001 caaa cab 010000001 caaaa");

int n = textBox2.Lines.Length;

}

private void Form1\_Load(object sender, EventArgs e)

{

}

private void Form1\_Load\_1(object sender, EventArgs e)

{

}

private void label5\_Click(object sender, EventArgs e)

{

}

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

{

textBox4.Clear();

uSyntAnalyzer Synt = new uSyntAnalyzer(SyntTree);

Synt.Lex.strPSource = textBox2.Lines;

Synt.Lex.strPMessage = textBox4.Lines;

Synt.Lex.enumPState = TState.Start;

Synt.Lex.hashTableRezerv = hashTableRezerv;

try

{

TreeNode parent = new TreeNode("Tree");

SyntTree.Nodes.Add(parent);

Synt.S(parent);

Synt.Lex.NextToken();

throw new Exception("Текст верный");

}

catch (Exception exc)

{

textBox4.Text += exc.Message;

textBox2.Select();

textBox2.SelectionStart = 0;

int n = 0;

for (int i = 0; i < Synt.Lex.intPSourceRowSelection; i++) n += textBox2.Lines[i].Length + 2;

n += Synt.Lex.intPSourceColSelection;

textBox2.SelectionLength = n;

}

}

private void button2\_Click(object sender, EventArgs e)

{

CLex Lex = new CLex();

Lex.strPSource = textBox2.Lines;

Lex.strPMessage = textBox4.Lines;

Lex.intPSourceColSelection = 0;

Lex.intPSourceRowSelection = 0;

int x = textBox2.TextLength;

int y = textBox2.Lines.Length;

textBox4.Text = "";

try

{

while (Lex.enumPState != TState.Finish)

{

Lex.NextToken();

string s1 = "", s = "";

switch (Lex.enumPToken)

{

case TToken.lxmIdentifier:

{

hashFunction.AddWord(hashTableIdentifier, Lex.strPLexicalUnit);

listBox4.Items.Add(Lex.strPLexicalUnit);

s1 = "id " + Lex.strPLexicalUnit;

break;

}

case TToken.lxmNumber:

{

hashFunction.AddWord(hashTableDigital, Lex.strPLexicalUnit);

listBox5.Items.Add(Lex.strPLexicalUnit);

s1 = "num " + Lex.strPLexicalUnit;

break;

}

case (TToken.lxmtz):

{

hashFunction.AddWord(hashTableRezerv, "\_");

listBox6.Items.Add(Lex.strPLexicalUnit);

s1 = "rez \_" + Lex.strPLexicalUnit;

break;

}

}

String m = "(" + s + "" + s1 + ")";

textBox4.Text += m;

}

}

catch (Exception exc)

{

textBox4.Text += exc.Message;

textBox2.Select();

textBox2.SelectionStart = 0;

int n = 0;

for (int i = 0; i < Lex.intPSourceRowSelection; i++) n += textBox2.Lines[i].Length + 2;

n += Lex.intPSourceColSelection;

textBox2.SelectionLength = n;

}

}

private void button3\_Click(object sender, EventArgs e)

{

if (hashFunction.SearchWord(hashTableIdentifier, listBox4.SelectedItem.ToString()) == 1)

{

button3.BackColor = Color.Green;

}

else

{

button3.BackColor = Color.Red;

}

}

private void button4\_Click(object sender, EventArgs e)

{

hashFunction.AddWord(hashTableIdentifier, textBox3.Text.ToString());

}

private void button5\_Click(object sender, EventArgs e)

{

if (hashFunction.RemoveWord(hashTableIdentifier, listBox4.SelectedItem.ToString()))

{

button5.BackColor = Color.Green;

}

else

{

button5.BackColor = Color.Red;

}

}

private void button6\_Click(object sender, EventArgs e)

{

if (hashFunction.RemoveWord(hashTableIdentifier, listBox4.SelectedItem.ToString()))

{

hashFunction.AddWord(hashTableIdentifier, textBox3.Text.ToString());

button6.BackColor = Color.Green;

}

else

{

button6.BackColor = Color.Red;

}

}

private void button7\_Click(object sender, EventArgs e)

{

listBox4.Items.Clear();

listBox5.Items.Clear();

listBox6.Items.Clear();

foreach (var entry in hashTableIdentifier)

{

listBox4.Items.Add(string.Join(", ", entry.Value));

}

foreach (var entry in hashTableDigital)

{

listBox5.Items.Add(string.Join(", ", entry.Value));

}

foreach (var entry in hashTableRezerv)

{

listBox6.Items.Add(string.Join(", ", entry.Value));

}

}

private void label7\_Click(object sender, EventArgs e)

{

}

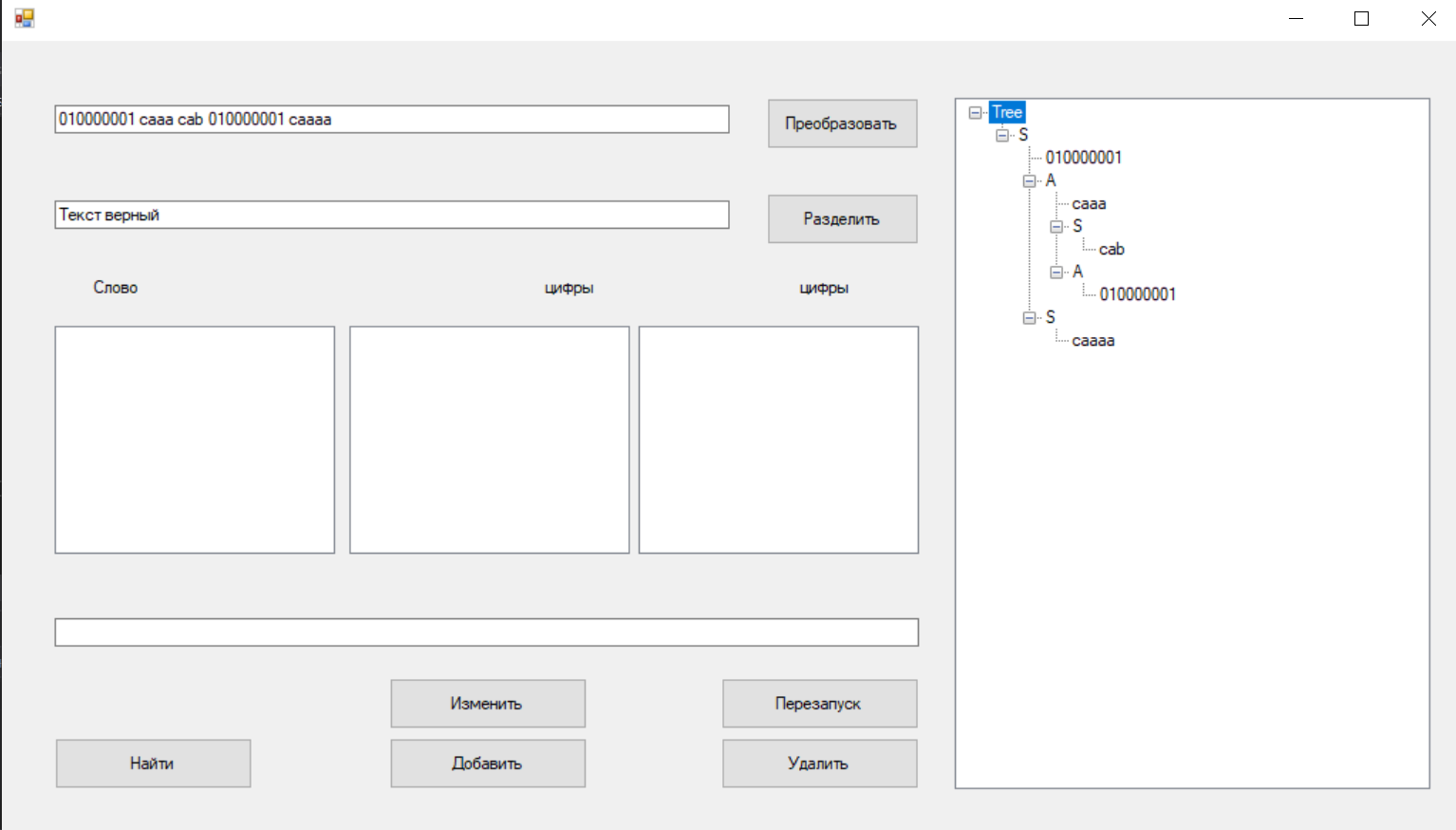
private void listBox6\_SelectedIndexChanged(object sender, EventArgs e)

{

}

}

}

Результат программы:  


**Лабораторная работа №6**

Выполнил: студент группы 4309

Курбанов Д.И.

В программировании есть инструмент, который позволяет экономить память и при этом обрабатывать огромные массивы данных. Это генераторы.

Генератор - это подпрограмма, которая может возвращать очередное значение и автоматически сохранять и возобновлять своё состояние для возврата следующего значения. Генератор похож на функцию, возвращающую массив, поскольку он имеет параметры, может быть вызван и возвращает последовательность значений.

Как выглядит цикл с генератором:

1. Цикл выполняется нужное количество раз.
2. На каждом шаге цикла генератор получает какое-то значение, отдаёт его в нужное место и забывает всё напрочь.
3. Генератор не помнит значение, которое он отдавал до этого, и не знает, что он будет отдавать на следующем шаге. Всё, что у него есть, — данные, которые нужно обработать на текущем шаге.
4. Память под работу генератора выделяется, только когда он генерирует новые данные. Пока генератор стоит или не выдаёт данные — память не выделяется.

Чаще всего генераторы используют как функции. Каждый раз, когда обращаются к такой функции-генератору, она делает так:

1. Берёт новую порцию данных из указанного ей источника.
2. Обрабатывает данные.
3. Возвращает результат.
4. Забывает про всё до следующего вызова.

Главный плюс генераторов — их можно указывать в качестве диапазона в циклах. На каждом шаге цикл получает новое значение от генератора и работает уже с ним. Как только у генератора заканчиваются варианты и он останавливается — цикл тоже останавливается.

Генератор — функция, которая генерирует последовательность результатов вместо одного значения. Вместо того, чтобы возвращать значение, мы создаём серию значений. Вызов функции генератора создает объект-генератор. Однако функция не запускается. Функция генератор выполняется только при вызове. Функция генератор возобновляется при следующем вызове. При завершении итератора возбуждается исключение.

Атрибутивная грамматика - это формальный способ дополнить формальную грамматику обработкой семантической информации. Семантическая информация хранится в атрибутах, связанных с терминальными и нетерминальными символами грамматики. Значения атрибутов являются результатом правил оценки атрибутов, связанных с созданием грамматики.

Атрибуты позволяют передавать информацию из любого места в абстрактном синтаксическом дереве в любое другое место контролируемым и формальным способом.

Каждая семантическая функция имеет дело с атрибутами символов, встречающимися только в одном производственном правиле: и параметры семантической функции, и ее результат являются атрибутами символов из одного конкретного правила. Когда семантическая функция определяет значение атрибута символа в левой части правила, атрибут называется синтезированным; в противном случае он называется унаследованным. Таким образом, синтезированные атрибуты служат для передачи семантической информации вверх по дереву синтаксического анализа, в то время как унаследованные атрибуты позволяют передавать значения от родительских узлов вниз и по всему синтаксическому дереву.

В простых приложениях, таких как вычисление арифметических выражений, атрибутивная грамматика может использоваться для описания всей задачи, которая должна быть выполнена, помимо прямого синтаксического анализа; в сложных системах, например, при создании средства языкового перевода, такого как компилятор, она может использоваться для проверки семантики, связанной с грамматикой, представляющей правила языка, которые явно не передаются определением синтаксиса. Она также может использоваться парсерами или компиляторами для перевода синтаксического дерева непосредственно в код для некоторой конкретной машины или на некоторый промежуточный язык.

**Задание**.

1. Перевести все числа в десятичное представление.

2. Выполнить вывод исходного текста в структурированном виде

Код программы:

TreeConverter.cs:

using System;

using System.Collections.Generic;

using System.Linq;

using System.Text;

using System.Threading.Tasks;

using System.Windows.Forms;

namespace WindowsFormsApp4

{

internal class TreeConverter

{

public void ConvertBinaryToDecimal(TreeView tree)

{

foreach (TreeNode node in tree.Nodes)

{

ConvertBinaryToDecimal(node);

}

}

private void ConvertBinaryToDecimal(TreeNode node)

{

if ((node.Text == "S") || (node.Text == "A"))

{

if (node.Nodes.Count > 0)

{

foreach (TreeNode node2 in node.Nodes)

{

string binaryValue = node2.Text;

if (IsBin(binaryValue))

{

int decimalValue = Convert.ToInt32(binaryValue, 2);

node2.Text = decimalValue.ToString();

}

}

}

}

foreach (TreeNode childNode in node.Nodes)

{

ConvertBinaryToDecimal(childNode);

}

}

private bool IsBin(string value)

{

foreach (var c in value)

if (c != '0' && c != '1')

return false;

return true;

}

}

}

Form1.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;

namespace WindowsFormsApp4

{

public partial class Form1 : Form

{

Dictionary<int, List<string>> hashTableIdentifier = new Dictionary<int, List<string>>();

Dictionary<int, List<string>> hashTableDigital = new Dictionary<int, List<string>>();

Dictionary<int, List<string>> hashTableRezerv = new Dictionary<int, List<string>>();

public MyHashFunction hashFunction = new MyHashFunction();

public Form1()

{

InitializeComponent();

//textBox2.AppendText("010000001 010000001 caaaa");

textBox2.AppendText("010000001 caaa cab 010000001 caaaa");

int n = textBox2.Lines.Length;

}

private void Form1\_Load(object sender, EventArgs e)

{

}

private void Form1\_Load\_1(object sender, EventArgs e)

{

}

private void label5\_Click(object sender, EventArgs e)

{

}

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

{

textBox4.Clear();

uSyntAnalyzer Synt = new uSyntAnalyzer(SyntTree);

Synt.Lex.strPSource = textBox2.Lines;

Synt.Lex.strPMessage = textBox4.Lines;

Synt.Lex.enumPState = TState.Start;

Synt.Lex.hashTableRezerv = hashTableRezerv;

try

{

TreeNode parent = new TreeNode("TreeView");

SyntTree.Nodes.Add(parent);

Synt.S(parent);

Synt.Lex.NextToken();

SyntTree.ExpandAll();

Generator.CopyTree(SyntTree, SyntTree2);

Generator.RefactorTree(SyntTree2);

SyntTree2.ExpandAll();

throw new Exception("Текст верный");

}

catch (Exception exc)

{

textBox4.Text += exc.Message;

textBox2.Select();

textBox2.SelectionStart = 0;

int n = 0;

for (int i = 0; i < Synt.Lex.intPSourceRowSelection; i++) n += textBox2.Lines[i].Length + 2;

n += Synt.Lex.intPSourceColSelection;

textBox2.SelectionLength = n;

}

}

private void button2\_Click(object sender, EventArgs e)

{

CLex Lex = new CLex();

Lex.strPSource = textBox2.Lines;

Lex.strPMessage = textBox4.Lines;

Lex.intPSourceColSelection = 0;

Lex.intPSourceRowSelection = 0;

int x = textBox2.TextLength;

int y = textBox2.Lines.Length;

textBox4.Text = "";

try

{

while (Lex.enumPState != TState.Finish)

{

Lex.NextToken();

string s1 = "", s = "";

switch (Lex.enumPToken)

{

case TToken.lxmIdentifier:

{

hashFunction.AddWord(hashTableIdentifier, Lex.strPLexicalUnit);

listBox4.Items.Add(Lex.strPLexicalUnit);

s1 = "id " + Lex.strPLexicalUnit;

break;

}

case TToken.lxmNumber:

{

hashFunction.AddWord(hashTableDigital, Lex.strPLexicalUnit);

listBox5.Items.Add(Lex.strPLexicalUnit);

s1 = "num " + Lex.strPLexicalUnit;

break;

}

case (TToken.lxmtz):

{

hashFunction.AddWord(hashTableRezerv, "\_");

listBox6.Items.Add(Lex.strPLexicalUnit);

s1 = "rez \_" + Lex.strPLexicalUnit;

break;

}

}

String m = "(" + s + "" + s1 + ")";

textBox4.Text += m;

}

}

catch (Exception exc)

{

textBox4.Text += exc.Message;

textBox2.Select();

textBox2.SelectionStart = 0;

int n = 0;

for (int i = 0; i < Lex.intPSourceRowSelection; i++) n += textBox2.Lines[i].Length + 2;

n += Lex.intPSourceColSelection;

textBox2.SelectionLength = n;

}

}

private void button3\_Click(object sender, EventArgs e)

{

if (hashFunction.SearchWord(hashTableIdentifier, listBox4.SelectedItem.ToString()) == 1)

{

button3.BackColor = Color.Green;

}

else

{

button3.BackColor = Color.Red;

}

}

private void button4\_Click(object sender, EventArgs e)

{

hashFunction.AddWord(hashTableIdentifier, textBox3.Text.ToString());

}

private void button5\_Click(object sender, EventArgs e)

{

if (hashFunction.RemoveWord(hashTableIdentifier, listBox4.SelectedItem.ToString()))

{

button5.BackColor = Color.Green;

}

else

{

button5.BackColor = Color.Red;

}

}

private void button6\_Click(object sender, EventArgs e)

{

if (hashFunction.RemoveWord(hashTableIdentifier, listBox4.SelectedItem.ToString()))

{

hashFunction.AddWord(hashTableIdentifier, textBox3.Text.ToString());

button6.BackColor = Color.Green;

}

else

{

button6.BackColor = Color.Red;

}

}

private void button7\_Click(object sender, EventArgs e)

{

listBox4.Items.Clear();

listBox5.Items.Clear();

listBox6.Items.Clear();

foreach (var entry in hashTableIdentifier)

{

listBox4.Items.Add(string.Join(", ", entry.Value));

}

foreach (var entry in hashTableDigital)

{

listBox5.Items.Add(string.Join(", ", entry.Value));

}

foreach (var entry in hashTableRezerv)

{

listBox6.Items.Add(string.Join(", ", entry.Value));

}

}

private void label7\_Click(object sender, EventArgs e)

{

}

private void listBox6\_SelectedIndexChanged(object sender, EventArgs e)

{

}

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

{

{

TreeConverter treeConverter = new TreeConverter();

treeConverter.ConvertBinaryToDecimal(SyntTree);

}

}

private void SyntTree2\_AfterSelect(object sender, TreeViewEventArgs e)

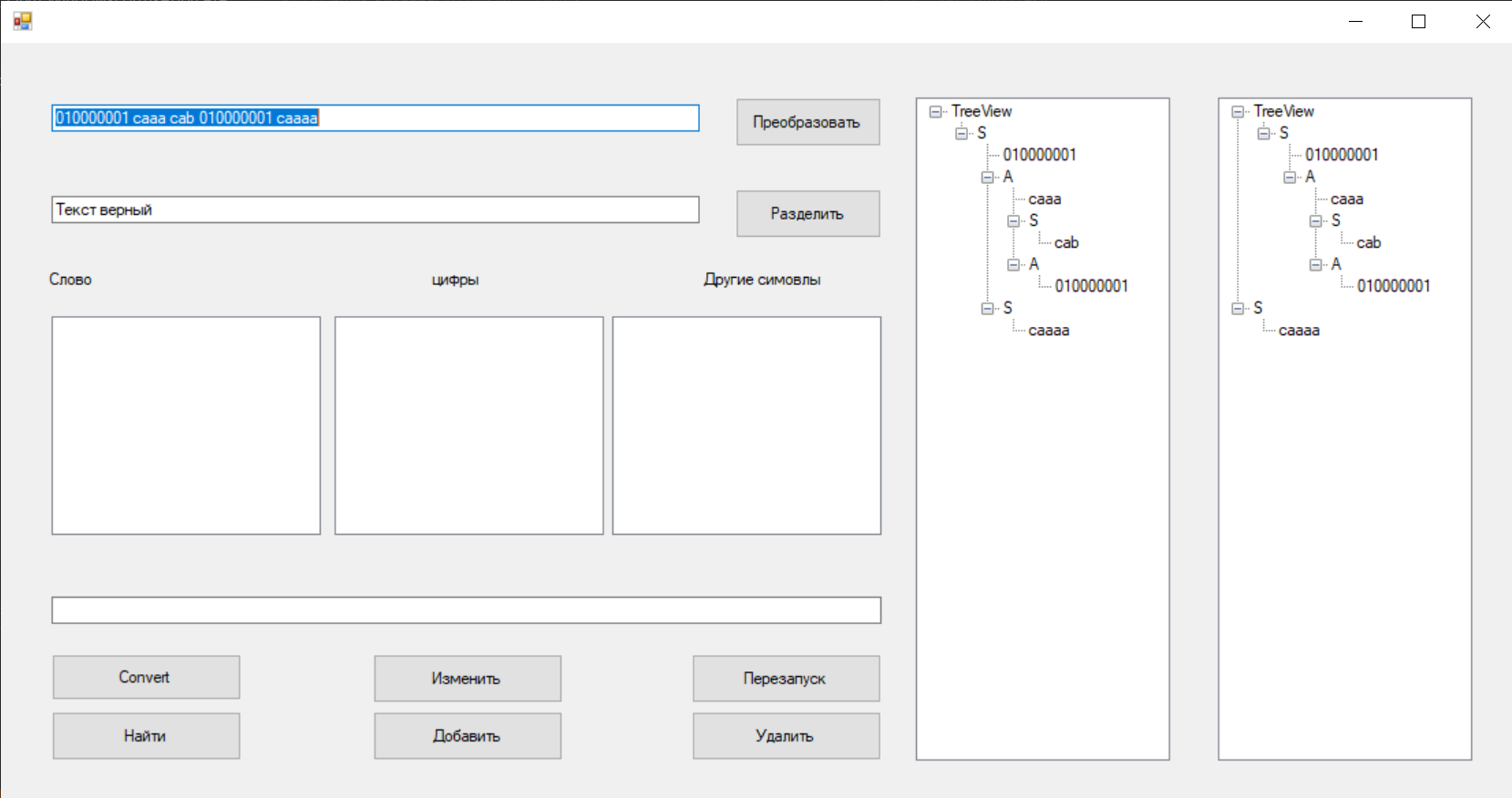
{

}

}

}

Результат работы программы:



**Лабораторная работа №7**

Выполнил: студент группы 4309

Курбанов Д.И.

Генератор - это подпрограмма, которая может возвращать очередное значение и автоматически сохранять и возобновлять своё состояние для возврата следующего значения. Генератор похож на функцию, возвращающую массив, поскольку он имеет параметры, может быть вызван и возвращает последовательность значений.

Как выглядит цикл с генератором:

1. Цикл выполняется нужное количество раз.
2. На каждом шаге цикла генератор получает какое-то значение, отдаёт его в нужное место и забывает всё напрочь.
3. Генератор не помнит значение, которое он отдавал до этого, и не знает, что он будет отдавать на следующем шаге. Всё, что у него есть, — данные, которые нужно обработать на текущем шаге.
4. Память под работу генератора выделяется, только когда он генерирует новые данные. Пока генератор стоит или не выдаёт данные — память не выделяется.

Чаще всего генераторы используют как функции. Каждый раз, когда обращаются к такой функции-генератору, она делает так:

1. Берёт новую порцию данных из указанного ей источника.
2. Обрабатывает данные.
3. Возвращает результат.
4. Забывает про всё до следующего вызова.

Главный плюс генераторов — их можно указывать в качестве диапазона в циклах. На каждом шаге цикл получает новое значение от генератора и работает уже с ним. Как только у генератора заканчиваются варианты и он останавливается — цикл тоже останавливается.

Генератор — функция, которая генерирует последовательность результатов вместо одного значения. Вместо того, чтобы возвращать значение, мы создаём серию значений. Вызов функции генератора создает объект-генератор. Однако функция не запускается. Функция генератор выполняется только при вызове. Функция генератор возобновляется при следующем вызове. При завершении итератора возбуждается исключение.

Атрибутивная грамматика - это формальный способ дополнить формальную грамматику обработкой семантической информации. Семантическая информация хранится в атрибутах, связанных с терминальными и нетерминальными символами грамматики. Значения атрибутов являются результатом правил оценки атрибутов, связанных с созданием грамматики.

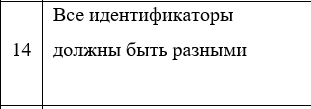
Генераторы в программировании представляют собой мощный инструмент, позволяющий работать с последовательностями данных. Однако, у генераторов есть свои ограничения и недостатки, которые могут затруднять их применение при решении сложных задач. Одно из ограничений генератора заключается в его однонаправленности. Генераторы позволяют передвигаться только вперед по последовательности данных, что ограничивает возможность обратного прохода или повторного прохода.

Атрибуты позволяют передавать информацию из любого места в абстрактном синтаксическом дереве в любое другое место контролируемым и формальным способом.

Каждая семантическая функция имеет дело с атрибутами символов, встречающимися только в одном производственном правиле: и параметры семантической функции, и ее результат являются атрибутами символов из одного конкретного правила. Когда семантическая функция определяет значение атрибута символа в левой части правила, атрибут называется синтезированным; в противном случае он называется унаследованным. Таким образом, синтезированные атрибуты служат для передачи семантической информации вверх по дереву синтаксического анализа, в то время как унаследованные атрибуты позволяют передавать значения от родительских узлов вниз и по всему синтаксическому дереву.

В простых приложениях, таких как вычисление арифметических выражений, атрибутивная грамматика может использоваться для описания всей задачи, которая должна быть выполнена, помимо прямого синтаксического анализа; в сложных системах, например, при создании средства языкового перевода, такого как компилятор, она может использоваться для проверки семантики, связанной с грамматикой, представляющей правила языка, которые явно не передаются определением синтаксиса. Она также может использоваться парсерами или компиляторами для перевода синтаксического дерева непосредственно в код для некоторой конкретной машины или на некоторый промежуточный язык.

**Задание**. Для предложенного преподавателем варианта контекстного условия расширить атрибутную грамматику из лабораторной работы № 4 добавлением атрибутов, правил их вычисления, правил вычисления контекстных условий. Включить в программу синтаксического анализатора из лабораторной работы № 4 действия по вычислению атрибутов и проверки контекстных условий.



Код программы:

Form1.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;

namespace WindowsFormsApp4

{

public partial class Form1 : Form

{

Dictionary<int, List<string>> hashTableIdentifier = new Dictionary<int, List<string>>();

Dictionary<int, List<string>> hashTableDigital = new Dictionary<int, List<string>>();

Dictionary<int, List<string>> hashTableRezerv = new Dictionary<int, List<string>>();

public MyHashFunction hashFunction = new MyHashFunction();

public Form1()

{

InitializeComponent();

//textBox2.AppendText("010000001 010000001 caaaa");

textBox2.AppendText("010000001 caa caaa 000 caaa");

int n = textBox2.Lines.Length;

}

private void Form1\_Load(object sender, EventArgs e)

{

}

private void Form1\_Load\_1(object sender, EventArgs e)

{

}

private void label5\_Click(object sender, EventArgs e)

{

}

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

{

if (SyntTree.Nodes.Count > 0)

{

SyntTree.Nodes.Clear();

SyntTree2.Nodes.Clear();

}

textBox4.Clear();

uSyntAnalyzer Synt = new uSyntAnalyzer(SyntTree);

Synt.Lex.strPSource = textBox2.Lines;

Synt.Lex.strPMessage = textBox4.Lines;

Synt.Lex.enumPState = TState.Start;

Synt.Lex.hashTableRezerv = hashTableRezerv;

try

{

TreeNode parent = new TreeNode("TreeView");

SyntTree.Nodes.Add(parent);

Synt.S(parent);

Synt.Lex.NextToken();

throw new Exception("Текст верный");

}

catch (Exception exc)

{

textBox4.Text += exc.Message;

textBox2.Select();

textBox2.SelectionStart = 0;

int n = 0;

for (int i = 0; i < Synt.Lex.intPSourceRowSelection; i++) n += textBox2.Lines[i].Length + 2;

n += Synt.Lex.intPSourceColSelection;

textBox2.SelectionLength = n;

}

Generator.CopyTree(SyntTree, SyntTree2);

Generator.RefactorTree(SyntTree2);

SyntTree.ExpandAll();

SyntTree2.ExpandAll();

}

private void button2\_Click(object sender, EventArgs e)

{

CLex Lex = new CLex();

Lex.strPSource = textBox2.Lines;

Lex.strPMessage = textBox4.Lines;

Lex.intPSourceColSelection = 0;

Lex.intPSourceRowSelection = 0;

int x = textBox2.TextLength;

int y = textBox2.Lines.Length;

textBox4.Text = "";

try

{

while (Lex.enumPState != TState.Finish)

{

Lex.NextToken();

string s1 = "", s = "";

switch (Lex.enumPToken)

{

case TToken.lxmIdentifier:

{

hashFunction.AddWord(hashTableIdentifier, Lex.strPLexicalUnit);

listBox4.Items.Add(Lex.strPLexicalUnit);

s1 = "id " + Lex.strPLexicalUnit;

break;

}

case TToken.lxmNumber:

{

hashFunction.AddWord(hashTableDigital, Lex.strPLexicalUnit);

listBox5.Items.Add(Lex.strPLexicalUnit);

s1 = "num " + Lex.strPLexicalUnit;

break;

}

case (TToken.lxmtz):

{

hashFunction.AddWord(hashTableRezerv, "\_");

listBox6.Items.Add(Lex.strPLexicalUnit);

s1 = "rez \_" + Lex.strPLexicalUnit;

break;

}

}

String m = "(" + s + "" + s1 + ")";

textBox4.Text += m;

}

}

catch (Exception exc)

{

textBox4.Text += exc.Message;

textBox2.Select();

textBox2.SelectionStart = 0;

int n = 0;

for (int i = 0; i < Lex.intPSourceRowSelection; i++) n += textBox2.Lines[i].Length + 2;

n += Lex.intPSourceColSelection;

textBox2.SelectionLength = n;

}

}

private void button3\_Click(object sender, EventArgs e)

{

if (hashFunction.SearchWord(hashTableIdentifier, listBox4.SelectedItem.ToString()) == 1)

{

button3.BackColor = Color.Green;

}

else

{

button3.BackColor = Color.Red;

}

}

private void button4\_Click(object sender, EventArgs e)

{

hashFunction.AddWord(hashTableIdentifier, textBox3.Text.ToString());

}

private void button5\_Click(object sender, EventArgs e)

{

if (hashFunction.RemoveWord(hashTableIdentifier, listBox4.SelectedItem.ToString()))

{

button5.BackColor = Color.Green;

}

else

{

button5.BackColor = Color.Red;

}

}

private void button6\_Click(object sender, EventArgs e)

{

if (hashFunction.RemoveWord(hashTableIdentifier, listBox4.SelectedItem.ToString()))

{

hashFunction.AddWord(hashTableIdentifier, textBox3.Text.ToString());

button6.BackColor = Color.Green;

}

else

{

button6.BackColor = Color.Red;

}

}

private void button7\_Click(object sender, EventArgs e)

{

listBox4.Items.Clear();

listBox5.Items.Clear();

listBox6.Items.Clear();

foreach (var entry in hashTableIdentifier)

{

listBox4.Items.Add(string.Join(", ", entry.Value));

}

foreach (var entry in hashTableDigital)

{

listBox5.Items.Add(string.Join(", ", entry.Value));

}

foreach (var entry in hashTableRezerv)

{

listBox6.Items.Add(string.Join(", ", entry.Value));

}

}

private void label7\_Click(object sender, EventArgs e)

{

}

private void listBox6\_SelectedIndexChanged(object sender, EventArgs e)

{

}

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

{

{

TreeConverter treeConverter = new TreeConverter();

treeConverter.ConvertBinaryToDecimal(SyntTree);

}

}

private void SyntTree2\_AfterSelect(object sender, TreeViewEventArgs e)

{

}

}

}

Generator.cs:

using System;

using System.Collections.Generic;

using System.Linq;

using System.Text;

using System.Threading.Tasks;

using System.Windows.Forms;

namespace WindowsFormsApp4

{

public class Generator

{

public static void CopyTree(TreeView orig, TreeView copy)

{

foreach (TreeNode node in orig.Nodes)

{

TreeNode copyNode = new TreeNode(node.Text);

copy.Nodes.Add(copyNode);

CopyNodes(node, copyNode);

}

}

public static void CopyNodes(TreeNode orig, TreeNode copy)

{

foreach (TreeNode node in orig.Nodes)

{

TreeNode copyNode = new TreeNode(node.Text);

copy.Nodes.Add(copyNode);

CopyNodes(node, copyNode);

}

}

public static void RefactorTree(TreeView tree)

{

foreach (TreeNode node in tree.Nodes)

{

RefactorTreeNodes(tree, node);

}

}

public static void RefactorTreeNodes(TreeView tree, TreeNode parent)

{

TreeNode nodeToMove = null;

foreach (TreeNode node in parent.Nodes)

{

if(parent.Text == "S" && node.Text == "S")

{

nodeToMove = node;

}

RefactorTreeNodes(tree, node);

}

if(nodeToMove != null)

{

parent.Nodes.Remove(nodeToMove);

tree.Nodes.Add(nodeToMove);

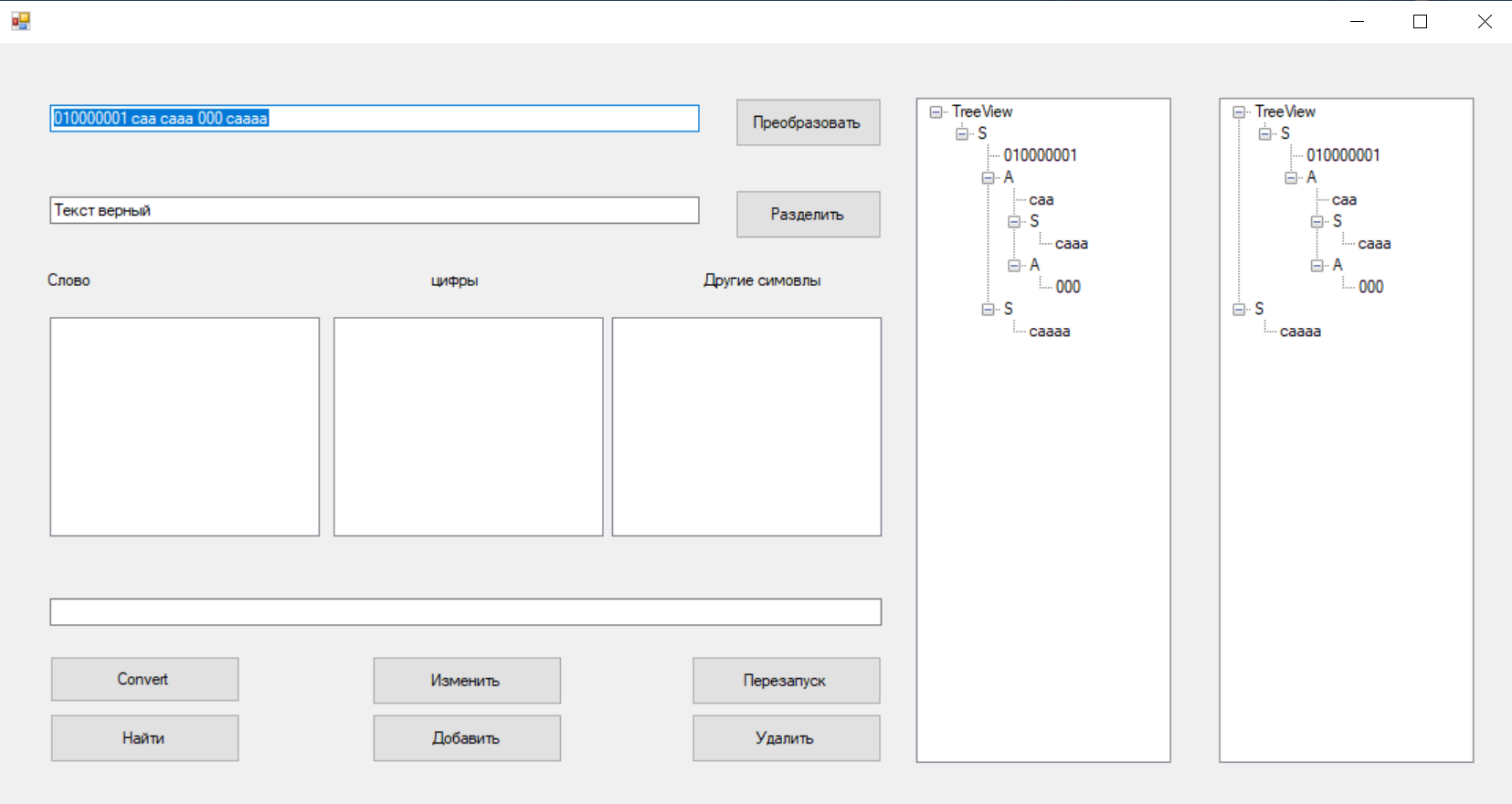
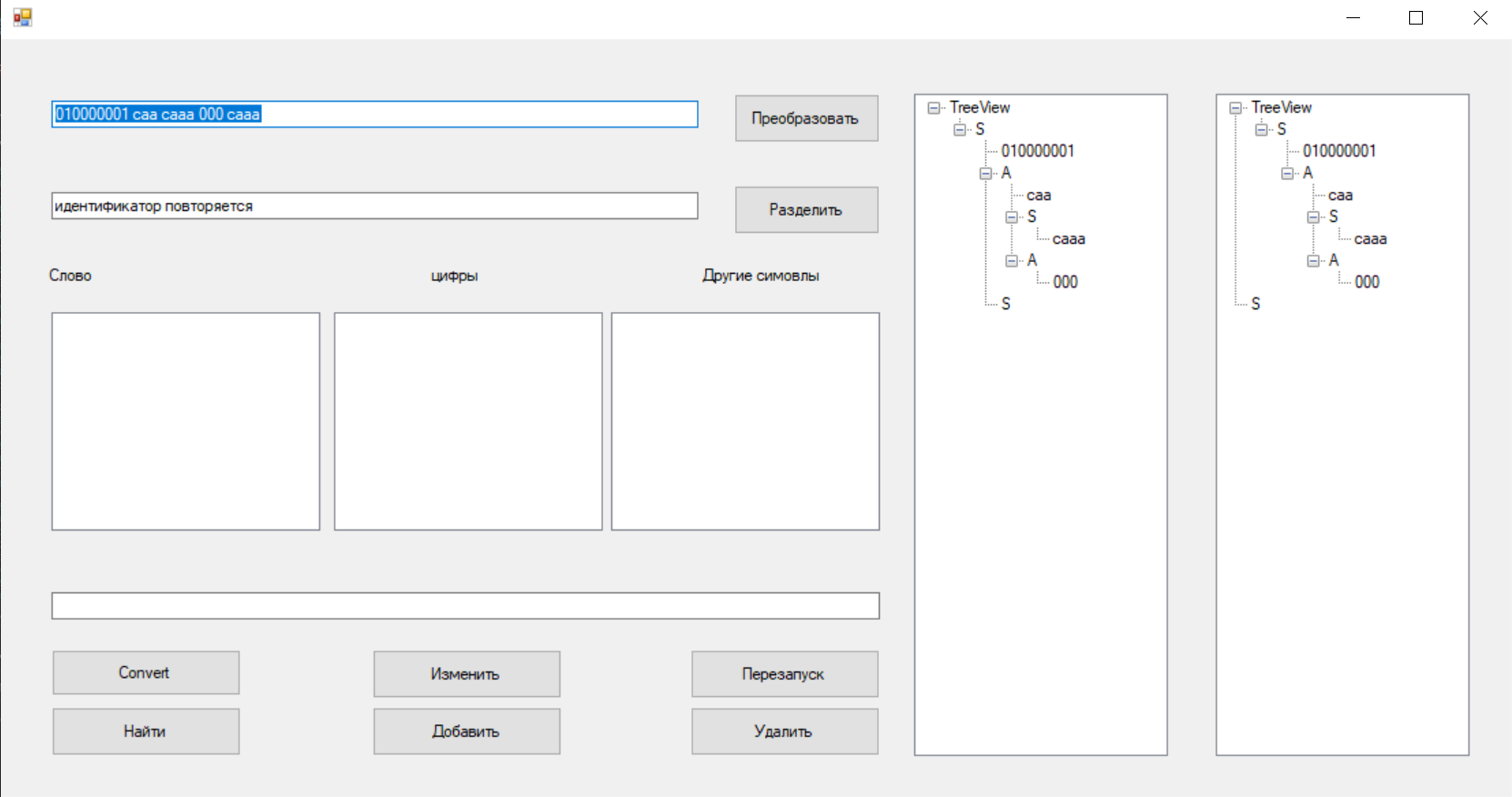
}

}

}

}

Результат работы программы:



**Лабораторная работа №8**

Выполнил: студент группы 4309

Курбанов Д.И.

Семантический анализатор: он использует синтаксическое дерево и таблицу символов, чтобы проверить, соответствует ли данная программа семантически определению языка. Он собирает информацию о типе и сохраняет ее либо в синтаксическом дереве, либо в таблице символов. Эта информация о типе впоследствии используется компилятором при генерации промежуточного кода.

Следующий шаг анализа текста программы – семантический, существенно отличается от синтаксического. И дело не столько в том, что фаза семантического анализа реализуется не формальными, а содержательными методами (т. е. на данный момент нет универсальных математических моделей и формальных средств описания «смысла» программы). Лексический и синтаксический анализ имеют дело со структурными, т. е. внешними, текстовыми конструкциями языка. Семантика же, ориентированная на содержательную интерпретацию, имеет дело с внутренним представлением «смысла» объектов, описанных в программе. Для любого, имеющего опыт практического программирования, ясно, что формальные конструкции языка дают описание свойств и действий над внутренними объектами, с которыми имеет дело программа. Для начала перечислим все, что их касается и лежит на поверхности:

· большинство объектов являются именованными. Имя объекта позволяет его идентифицировать, существуют различные области действия имен, соглашения об именах, различные умолчания и т. п.. Все это относится к семантике;

· виды, сложность и набор характеристик объектов различаются в разных языках программирования и сильно зависят от области приложения языка (в этом смысле семантика языков программирования более разнообразна, нежели синтаксис и лексика). Например, классический Си, ориентированный на максимальное приближение к архитектуре компьютера, работает с такими объектами, как типы данных, переменные, функции. Все они имеют различные свойства и характеристики. Например, переменная характеризуется именем, типом данных, размерностью, областью действия, временем жизни, текущим значением;

· объекты связаны между собой (ссылаются друг на друга). В том же Си переменная ссылается на описание того типа данных, к которому она относится, далее производный тип данных ссылается на базовый и т. п.. Можно сказать, что семантика программы во внутреннем представлении выглядит как система взаимосвязанных объектов;

· внутреннее представление семантики программы не совсем удачно называется семантическими таблицами. На самом деле структура данных, соответствующая представлению семантики, может быть любой. Термин «таблицы» говорит о том, что имеются множества объектов различных типов, для каждого из которых заведена отдельная таблица, но нельзя забывать, что элементы различных таблиц связаны между собой. Наиболее близкий термин для описания подобной системы – база данных.

Семантика программы – внутренняя модель (база данных) множества именованных объектов, с которыми работает программа, с описанием их свойств, характеристик и связей.

Семантика программы тоже не обладает структурной целостностью и представлена фрагментарно, но при этом связана с синтаксисом следующим образом:

· один и тот же семантический объект (например, переменная) может встречаться в различных, синтаксически несвязанных частях программы;

· синтаксические конструкции описаний, определений и объявлений являются источником семантики объектов программы, они «заявляют» о существовании объектов и задают их свойства;

· синтаксические конструкции, связанные с действиями, выполняемыми над объектами, являются потребителями семантики, их интерпретация, корректность, «смысл» зависят от семантических свойств объекта. Забегая вперед, можно заметить, что заключительная фаза трансляции (генерация кода, интерпретация) может рассматриваться как особые семантические действия, производимые над объектами;

· первичным источником семантики является лексический анализ. Некоторые из лексем (например, идентификаторы и константы) наряду с классом лексемы (обозначение выходной единицы лексического анализа), т. е. символом (обозначение той же единицы на входе синтаксического анализатора) имеют значение. Значением лексемы является сама распознанная цепочка литер, она и представляет семантическую составляющую лексемы, которая попадает в семантические таблицы;

· лексемы, или то же самое, что терминальные символы входной строки (в терминах синтаксического анализа), ссылаются в семантические таблицах на свою семантику. В дальнейшем каждый промежуточный нетерминал также ссылается на собственную семантику. При этом любое правило преобразует семантику терминалов и нетерминалов правой части в семантику нетерминала левой части при помощи назначенной правилу семантической процедуры. Таким образом, формирование семантической составляющей связано с движением снизу вверх по синтаксическому дереву, от вершин – потомков к предкам.

В процессе семантического анализа проверяется наличие семантических ошибок в исходной программе и накапливается информация о типах для следующей стадии – генерации кода. При семантическом анализе используются иерархические структуры, полученные во время синтаксического анализа для идентификации операторов и операндов выражений и инструкций.

Важным аспектом семантического анализа является проверка типов, когда компилятор проверяет, что каждый оператор имеет операнды допустимого спецификациями языка типа. Например, определение многих языков программирования требует, чтобы при использовании действительного числа в качестве индекса массива генерировалось сообщение об ошибке. В то же время спецификация языка может позволить определенное преобразование типов, например, когда бинарный арифметический оператор применяется к операндам целого и действительного типов. В этом случае компилятору может потребоваться преобразование целого числа в действительное.

Все типы известны во время компиляции, и это относится к типам выражений, идентификаторам и литералам. При этом неважно, насколько сложным является выражение: его тип может определяться во время компиляции за определенное количество шагов исходя из типов его составляющих. Фактически, это позволяет производить контроль типов во время компиляции и находить в процессе компиляции многие программные ошибки.

В процессе компиляции анализатору требуются таблицы:

– таблица символов;

– таблица типов;

– таблица функций;

– таблица меток.

Основная задача таблицы символов – установление соответствия между переменной и ее типом. Операции, связанные с таблицей символов:

– операция, соответствующая определяющему вхождению переменной. В таблицу символов помещаются имя переменной и ее тип;

– операция, соответствующая применимому вхождению переменной. Исследуется таблица символов для нахождения типа переменной.

Как известно, один и тот же синтаксис может быть представлен различными грамматиками. Они эквивалентны между собой, поскольку распознают одно и то же множество цепочек (Как известно, проблема эквивалентности формальных грамматик алгоритмически неразрешима, т. е. невозможно разработать алгоритм, устанавливающий эквивалентность двух произвольных грамматик). Мы уже видели, что требования к грамматикам восходящего и нисходящего разбора также различны. Различны и технологически подходы к реализации одних и тех же элементов синтаксиса (повторений, необязательных элементов синтаксиса и т. п.). Поэтому получается, что в различных грамматиках при разборе одного и того же предложении будут строиться различные деревья, но результат трансляции (например, последовательность выполняемых операций в выражении) должен быть если не одинаковым, то эквивалентным(разобрать методы и описать свой, задание написать).

Ошибки, распознанные семантическим анализатором, заключаются в следующем:

1. Несоответствие типов
2. Необъявленные переменные
3. Неправильное использование идентификатора

В данной лабораторной работе был использован третий пункт, я прописал условия для идентификатора, при неправильном использовании которых выдавало сообщение об ошибке

Функции семантического анализа:

1. Проверка типов – Гарантирует, что типы данных используются в соответствии их определением.
2. Проверка меток – Программа должна содержать ссылки на ярлыки.
3. Проверка управления потоком – Проверяет, что управляющие структуры используются надлежащим образом.

**Задание**. Создать семантический анализатор.

Код программы:

uSemantAnalyzer.cs:

using System;

using System.Collections.Generic;

using System.Drawing;

using System.Linq;

using System.Text;

using System.Threading.Tasks;

using System.Windows.Forms;

namespace WindowsFormsApp4

{

class uSemantAnalyzer

{

public int i = 0;

public string strIndentifier;

public string strDigital;

private TreeView tree;

public uSemantAnalyzer()

{

}

public uSemantAnalyzer(TreeView treeView)

{

tree = treeView;

TreeController(tree);

}

public void TreeController(TreeView tree)

{

foreach (TreeNode node in tree.Nodes)

{

TreeController(node);

}

}

public void TreeController(TreeNode node)

{

if (node.Text == "A")

{

if (node.Nodes.Count > 1)

{

strIndentifier = node.Nodes[1].Nodes[0].Text.ToString();

strDigital = node.Nodes[2].Nodes[0].Text.ToString();

Check(strIndentifier, strDigital, node);

}

}

foreach (TreeNode childNode in node.Nodes)

{

TreeController(childNode);

}

}

private void Check(string ident, string digit, TreeNode node)

{

int identLen = ident.Length;

if (ident.Count(x => x == 'a') < digit.Count(x => x == '0'))

{

tree.SelectedNode = node;

tree.SelectedNode.BackColor = Color.Red;

throw new Exception("Условие не выполнено");

}

}

}

}

Form1.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;

namespace WindowsFormsApp4

{

public partial class Form1 : Form

{

Dictionary<int, List<string>> hashTableIdentifier = new Dictionary<int, List<string>>();

Dictionary<int, List<string>> hashTableDigital = new Dictionary<int, List<string>>();

Dictionary<int, List<string>> hashTableRezerv = new Dictionary<int, List<string>>();

public MyHashFunction hashFunction = new MyHashFunction();

public Form1()

{

InitializeComponent();

//textBox2.AppendText("010000001 010000001 caaaa");

textBox2.AppendText("010000001 ca caa 000 caaa");

int n = textBox2.Lines.Length;

}

private void Form1\_Load(object sender, EventArgs e)

{

}

private void Form1\_Load\_1(object sender, EventArgs e)

{

}

private void label5\_Click(object sender, EventArgs e)

{

}

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

{

if (SyntTree.Nodes.Count > 0)

{

SyntTree.Nodes.Clear();

SyntTree2.Nodes.Clear();

}

textBox4.Clear();

uSyntAnalyzer Synt = new uSyntAnalyzer(SyntTree);

Synt.Lex.strPSource = textBox2.Lines;

Synt.Lex.strPMessage = textBox4.Lines;

Synt.Lex.enumPState = TState.Start;

Synt.Lex.hashTableRezerv = hashTableRezerv;

try

{

TreeNode parent = new TreeNode("TreeView");

SyntTree.Nodes.Add(parent);

Synt.S(parent);

Synt.Lex.NextToken();

throw new Exception("Текст верный");

}

catch (Exception exc)

{

textBox4.Text += exc.Message;

textBox2.Select();

textBox2.SelectionStart = 0;

int n = 0;

for (int i = 0; i < Synt.Lex.intPSourceRowSelection; i++) n += textBox2.Lines[i].Length + 2;

n += Synt.Lex.intPSourceColSelection;

textBox2.SelectionLength = n;

}

Generator.CopyTree(SyntTree, SyntTree2);

Generator.RefactorTree(SyntTree2);

SyntTree.ExpandAll();

SyntTree2.ExpandAll();

}

private void button2\_Click(object sender, EventArgs e)

{

CLex Lex = new CLex();

Lex.strPSource = textBox2.Lines;

Lex.strPMessage = textBox4.Lines;

Lex.intPSourceColSelection = 0;

Lex.intPSourceRowSelection = 0;

int x = textBox2.TextLength;

int y = textBox2.Lines.Length;

textBox4.Text = "";

try

{

while (Lex.enumPState != TState.Finish)

{

Lex.NextToken();

string s1 = "", s = "";

switch (Lex.enumPToken)

{

case TToken.lxmIdentifier:

{

hashFunction.AddWord(hashTableIdentifier, Lex.strPLexicalUnit);

listBox4.Items.Add(Lex.strPLexicalUnit);

s1 = "id " + Lex.strPLexicalUnit;

break;

}

case TToken.lxmNumber:

{

hashFunction.AddWord(hashTableDigital, Lex.strPLexicalUnit);

listBox5.Items.Add(Lex.strPLexicalUnit);

s1 = "num " + Lex.strPLexicalUnit;

break;

}

case (TToken.lxmtz):

{

hashFunction.AddWord(hashTableRezerv, "\_");

listBox6.Items.Add(Lex.strPLexicalUnit);

s1 = "rez \_" + Lex.strPLexicalUnit;

break;

}

}

String m = "(" + s + "" + s1 + ")";

textBox4.Text += m;

}

}

catch (Exception exc)

{

textBox4.Text += exc.Message;

textBox2.Select();

textBox2.SelectionStart = 0;

int n = 0;

for (int i = 0; i < Lex.intPSourceRowSelection; i++) n += textBox2.Lines[i].Length + 2;

n += Lex.intPSourceColSelection;

textBox2.SelectionLength = n;

}

}

private void button3\_Click(object sender, EventArgs e)

{

if (hashFunction.SearchWord(hashTableIdentifier, listBox4.SelectedItem.ToString()) == 1)

{

button3.BackColor = Color.Green;

}

else

{

button3.BackColor = Color.Red;

}

}

private void button4\_Click(object sender, EventArgs e)

{

hashFunction.AddWord(hashTableIdentifier, textBox3.Text.ToString());

}

private void button5\_Click(object sender, EventArgs e)

{

if (hashFunction.RemoveWord(hashTableIdentifier, listBox4.SelectedItem.ToString()))

{

button5.BackColor = Color.Green;

}

else

{

button5.BackColor = Color.Red;

}

}

private void button6\_Click(object sender, EventArgs e)

{

if (hashFunction.RemoveWord(hashTableIdentifier, listBox4.SelectedItem.ToString()))

{

hashFunction.AddWord(hashTableIdentifier, textBox3.Text.ToString());

button6.BackColor = Color.Green;

}

else

{

button6.BackColor = Color.Red;

}

}

private void button7\_Click(object sender, EventArgs e)

{

listBox4.Items.Clear();

listBox5.Items.Clear();

listBox6.Items.Clear();

foreach (var entry in hashTableIdentifier)

{

listBox4.Items.Add(string.Join(", ", entry.Value));

}

foreach (var entry in hashTableDigital)

{

listBox5.Items.Add(string.Join(", ", entry.Value));

}

foreach (var entry in hashTableRezerv)

{

listBox6.Items.Add(string.Join(", ", entry.Value));

}

}

private void label7\_Click(object sender, EventArgs e)

{

}

private void listBox6\_SelectedIndexChanged(object sender, EventArgs e)

{

}

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

{

{

TreeConverter treeConverter = new TreeConverter();

treeConverter.ConvertBinaryToDecimal(SyntTree);

}

}

private void SyntTree2\_AfterSelect(object sender, TreeViewEventArgs e)

{

}

}

}

uSyntAnalyzer.cs:

using System;

using System.Collections.Generic;

using System.Linq;

using System.Text;

using System.Threading.Tasks;

using System.Windows.Forms;

namespace WindowsFormsApp4

{

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();

TToken first;

bool init;

TreeView tree;

public uSyntAnalyzer(TreeView treeView)

{

tree = treeView;

}

public void S(TreeNode highParent)

{

TreeNode parent = new TreeNode("S");

highParent.Nodes.Add(parent);

Lex.NextToken();

if (Lex.enumPToken == TToken.lxmNumber)

{

parent.Nodes.Add(new TreeNode(Lex.strPLexicalUnit));

Lex.NextToken();

A(parent);

S(parent);

}

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

{

if (IsIdentifierUnique(Lex.strPLexicalUnit))

{

parent.Nodes.Add(new TreeNode(Lex.strPLexicalUnit));

Lex.NextToken();

}

else throw new Exception("идентификатор повторяется");

}

else throw new Exception("Ожидался число или идентификатор");

}

public void A(TreeNode highParent)

{

TreeNode parent = new TreeNode("A");

highParent.Nodes.Add(parent);

Lex.NextToken();

if (Lex.enumPToken == TToken.lxmIdentifier)

{

if (IsIdentifierUnique(Lex.strPLexicalUnit))

{

parent.Nodes.Add(new TreeNode(Lex.strPLexicalUnit));

Lex.NextToken();

S(parent);

A(parent);

uSemantAnalyzer uSemant = new uSemantAnalyzer(tree);

}

else throw new Exception("идентификатор повторяется");

}

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

{

parent.Nodes.Add(new TreeNode(Lex.strPLexicalUnit));

Lex.NextToken();

}

else throw new Exception("Ожидался число или идентификатор");

}

private bool IsIdentifierUnique(string identifier)

{

foreach (TreeNode node in tree.Nodes)

{

if (!IsIdentifierUniqueInNode(identifier, node))

{

return false;

}

}

return true;

}

private bool IsIdentifierUniqueInNode(string identifier, TreeNode node)

{

if (node.Text == identifier)

{

return false;

}

foreach (TreeNode child in node.Nodes)

{

if (!IsIdentifierUniqueInNode(identifier, child))

{

return false;

}

}

return true;

}

}

}

Результат работы программы: