МИНИСТЕРСТВО ОБРАЗОВАНИЯ И НАУКИ РОССИЙСКОЙ ФЕДЕРАЦИИ

Федеральное государственное бюджетное образовательное учреждение

высшего образования

УЛЬЯНОВСКИЙ ГОСУДАРСТВЕННЫЙ ТЕХНИЧЕСКИЙ УНИВЕРСИТЕТ

кафедра «Вычислительная техника».

**Курсовой проект**

по дисциплине: «Автоматизация проектирования технических средств»

Выполнил:

студент 4 курса, гр. ИВТВМбд-41

Захарычев Никита Алексеевич.

Проверил:

кандидат технических наук,

доцент кафедры ВТ

Войт Николай Николаевич.

г. Ульяновск, 2018

# Оглавление

[Оглавление 2](#_Toc503435372)

[Введение 3](#_Toc503435373)

[Постановка задачи 4](#_Toc503435374)

[Теоретические сведения 4](#_Toc503435375)

[1. Логическая схема алгоритма (ЛСА) 4](#_Toc503435376)

[2. Автомат с магазинной памятью 6](#_Toc503435377)

[Описание протокола входных данных 7](#_Toc503435378)

[Описание выходных данных 7](#_Toc503435379)

[Программная реализация 8](#_Toc503435380)

[Описание алгоритма разбора ЛСА 9](#_Toc503435381)

[Визуализация автомата 10](#_Toc503435382)

[Тестирование 11](#_Toc503435383)

[Заключение 13](#_Toc503435384)

[Список литературы 14](#_Toc503435385)

[Приложение. Исходный код 15](#_Toc503435386)

# Введение

В настоящее время особое место среди информационных технологий занимает автоматизация проектирования. Все современные конструкторские бюро, офисы, проектные подразделения насыщены вычислительной техникой. Предприятия, ведущие разработки без САПР оказываются неконкурентоспособными как из-за больших материальных и временных затрат на проектирование, так и из-за невысокого качества проектов. Поэтому, знание основ автоматизации проектирования и умение работать со средствами САПР требуется практически любому инженеру-разработчику. Автоматизация проектирования как дисциплина является составной, ее частями являются многие другие современные информационные технологии. Так, техническое обеспечение систем автоматизированного проектирования (САПР) основано на использовании телекоммуникационных технологий и вычислительных сетей, в САПР используются персональные компьютеры и рабочие станции. Программные комплексы САПР относятся к числу наиболее сложных современных программных систем, основанных на различных операционных системах, языках программирования и других, современных технологиях, реляционных и объектно-ориентированных системах управления базами данных (СУБД), стандартах открытых систем и обмена данными в компьютерных средах. [1]

Целью данного курсового проекта является разработка приложения, представляющего инструмент по обработке частичного микропрограммного автомата.

# Постановка задачи

Задачей данного курсового проекта является разработать программное средство для анализа логической схемы алгоритма и визуализации алгоритма в виде графа.

Полное техническое задание включает в себя:

1. Разработка парсера для обработки входных данных
2. Разработка структуры для хранения состояний, переходов и условий автомата
3. Разработка модуля для проверки переходов на основе автомата с магазинной памятью
4. Разработка и анализ протокола выходных данных
5. Разработка средства для визуализации алгоритма

# Теоретические сведения

# Логическая схема алгоритма (ЛСА)

Логическая схема алгоритма (ЛСА) является одним из основных способов задания закона функционирования автоматов. Формальный язык для описания алгоритмов был первоначально ориентирован на задачи оптимизации программ для ЭВМ, а в последствии оказался удобным и для оптимизации микропрограмм. Задачи этих классов широко распространены как в области проектирования средств вычислительной техники, так и при построении систем автоматического управления, связи и многих других.

Большим плюсом логической схемы алгоритма является компактность и возможность легкого преобразования в другую из форм задания автоматов.

Реализация алгоритма есть последовательное выполнение команд ЭВМ, каждая из которых в свою очередь является последовательностью элементарных действий-микрокоманд, выполняемых за один машинный такт.

Основное достоинство рассматриваемых ниже логических схем алгоритмов (ЛСА) состоит в том, что, являясь по существу разновидностью языка операторных схем, они допускают запись алгоритма в строчку, что часто является удобным, т.к. появляется возможность исключить процесс рисования, вычерчивания, как это имеет место в ГСА. Важным является также наличие развитой системы преобразований ЛСА и возможности формального перехода к автоматному отображению.

Основными элементами ЛСА являются так же, как и в ГСА, операторы и логические условия.

Основные отличия от ГСА состоят в том, что для указания взаимосвязей между операторами и логическими условиями используются верхние и нижние стрелки.

Логической схемой алгоритма называется строчка, составленная из символов операторов А_0,А_1,\dots,А_n,A_k, или Y_0, Y_1,\dots,Y_k и логических условий \alpha_{ji}, а также верхних и нижних стрелок. Иногда верхние и нижние стрелки заменяют на правые и левые полускобки.

ЛСА- строчка, составленная из символов операторов Y_0, Y_1,\dots,Y_k, логических условий xi и верхних \uparrow и нижних \downarrow стрелок, причем:

* Сильная операторная вершина Y_0(Y_н) и одна конечная Y_k ;
* Строка начинается с Y_0 и заканчивается Y_k ;
* Не должно быть двух нижних стрелок \downarrow с одинаковыми номерами;
* Для каждой нижней стрелки \downarrow должна быть по крайней мере одна верхняя;

Переход по логическому условию X_i, стоящему в ЛСА

\dots Y_p X_i \uparrow Y_m \dots  \downarrow Y_n

осуществляется так:

Если x_i=1, то после Y_p выполнится Y_n.

Если x_i=0, то после Y_p выполнится Y_m

Безусловный переход для ясности может быть обозначен дополнительным символом, например \omega \uparrow.[2]

# Автомат с магазинной памятью

Автоматы с магазинной памятью, или магазинные автоматы (сокращенно — МП-автоматы), образуют класс распознающих моделей для КС-языков точно так же, как конечные автоматы являются распознающими моделями в классе регулярных языков. МП-автомат—это конечный автомат, снабженный дополнительным стеком (магазином) для хранения промежуточной информации потенциально бесконечного объема (рис. 1). МП-автомат имеет конечное множество внутренних состояний S с начальным состоянием s0 и подмножеством F заключительных состояний, конечный алфавит входных сигналов X и конечный алфавит магазина Г с маркером ┴, который обозначает дно стека. Функция переходов 8 по каждой тройке <текущее внутреннее состояние, очередной входной сигнал, верхний символ магазина> определяет на очередном шаге функционирования следующее состояние и цепочку символов магазинной памяти, записываемых в магазин вместо прочитанного верхнего символа (условимся, что цепочка записывается в стек «хвостом вперед», т. е. сначала в стек записывается последний символ цепочки, затем предпоследний и т. д.). МП-автомат распознает входную цепочку, если после ее завершения на входе автомат перейдет в одно из заключительных состояний и его магазин будет пустым. [3]



**Рисунок 1. МП-автомат**

# Описание протокола входных данных

На вход программы подается строка, содержащая логическую схему алгоритма следующего вида:

Y0 ↓4 Y1 X1 ↑1 X2 ↑2 Y5 W ↑3 ↓1 Y2 ↓2 Y3 X3 ↑4 Y4 ↓3 Y6

Где каждый оператор отделен от других символом пробела.

Y – оператор, обозначающий состояние, каждое состояние должно иметь уникальный номер.

X – оператор условия. Возможно использование повторяющихся номеров для одинаковых условий.

W – оператор безусловного перехода

↑ - оператор начала перехода

↓ - оператор конца перехода. Должен быть уникальным

Иногда верхние и нижние стрелки заменяют на правые и левые полускобки.

# Описание выходных данных

Выходные данные представлены в следующем формате:

Y0:

Y1 :

Y1:

Y3 : X1

Y2 : !X1

Y2:

Y4 : X2

Y3 : !X2

Y3:

Y3 : X2X1

Y2 : X2!X1

Y4 : !X2

Здесь для каждого состояния выводится его номер и все возможные состояния, которые достижимы из него по соответствующим условиям перехода.

# Программная реализация

Программа реализует требуемый функционал таким образом:

* считывание описания автомата
* формирование набора переходов
* анализ набора переходов путем использования мп-автоматной модели
* визуальное отображение структуры алгоритма

Далее запускается итеративный алгоритм, выполняющий на каждой итерации:

* определение текущего состояния
* проверка на зацикливание (если автомат уже был в данном состоянии алгоритм завершается)
* проверка возможности совершения перехода (если отсутствуют переходы из данного состояния или они невозможны алгоритм завершается)
* нахождения возможного перехода
* совершение перехода
* запись перехода в список возможных переходов из текущего состояния
* перевод автомата в новое состояние

По окончании итеративного алгоритма выводится итоговое состояние автомата, список переходов и условий переходов между состояниями.

Программа реализована с применением языка программирования С# и платформы Microsoft .NET Framework. Для построения графического интерфейса использовалась технология Windows Forms.

# Описание алгоритма разбора ЛСА

Считывание входной строки (ЛСА) с консоли:

Console.WriteLine("Введите входную строку (ЛСА):");

input = Console.ReadLine();

После необходимо «отделить» переменные, что относится к X, Y и стрелки переходов (в данном случаем будем использовать полу скобки). Для это будем использовать следующую функцию:

public static void findXY()

Также необходимо получить позицию каждого элемента в строке. Для этого используем функцию public static int getNumber.

Для хранения состояния и условия перехода в него используется класс StateLSA.

Для поиска стрелок(полу скобок) используется функция: findRight.

Функция public static void addCond\_if(List<StateLSA> list, string cond\_if) используется для инициализации условии переходов в состояние.

Внутреннее представление ЛСА было реализовано в виде списка смежности.

public static List<List<StateLSA>> graph = new List<List<StateLSA>>();

В этом списке для каждой вершины определен список состояний, в которые можно попасть из нее и условия перехода.

Разбор строки ЛСА выполняется рекурсивно с помощью функции public static void getLSA() которая возвращает все доступные из заданной позиции состояния.

Начиная с указанной позиции в строке ЛСА запускается цикл по всем следующим символам строки. Среди входной строки ищется состояние (Y), для него создается экземпляр класса StateLSA, который определяет состояние и условие перехода. При встрече условия перехода (X), выполняется поиск стрелок (полускобок), имеющих одинаковые индексы (номера). В функцию getState передается позиция вершины, для которой мы хотим найти все доступные из нее другие вершины, функция запускается рекурсивно с позициями в строке, после верхней стрелки – если условие ложное и после нижней стрелки – если условие истинно. При встрече символа “W” выполняется переход только к нижней стрелке. Также реализован поиск ошибок с непосредственным выводом в месте ошибки.

# Визуализация автомата

Для визуализации автомата были реализованы следующие функции:

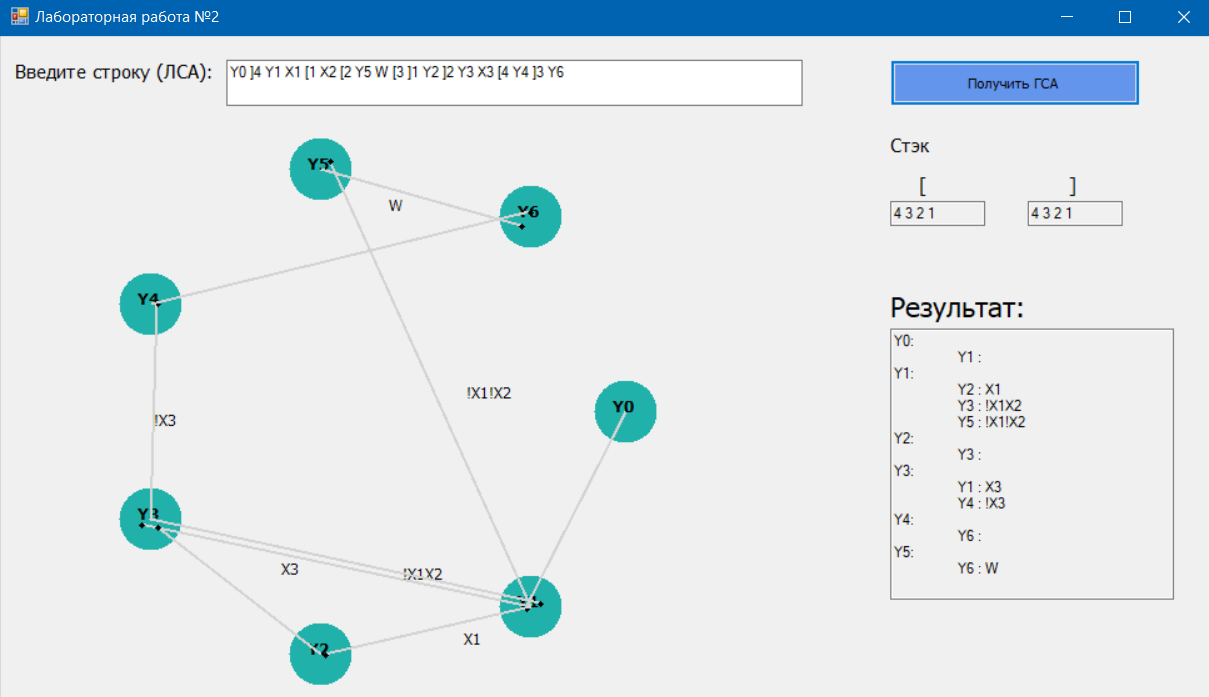
* protected override void OnPaint(PaintEventArgs e) – функция отрисовки состояний и переходов
* void getState() – функция создания состояния
* void getGSA() – функция, реализующая получение ГСА

Также реализовано 2 класса (State и Line), которые представляют из себя создание и отрисовки состояний(кругов) и переходов (линий).

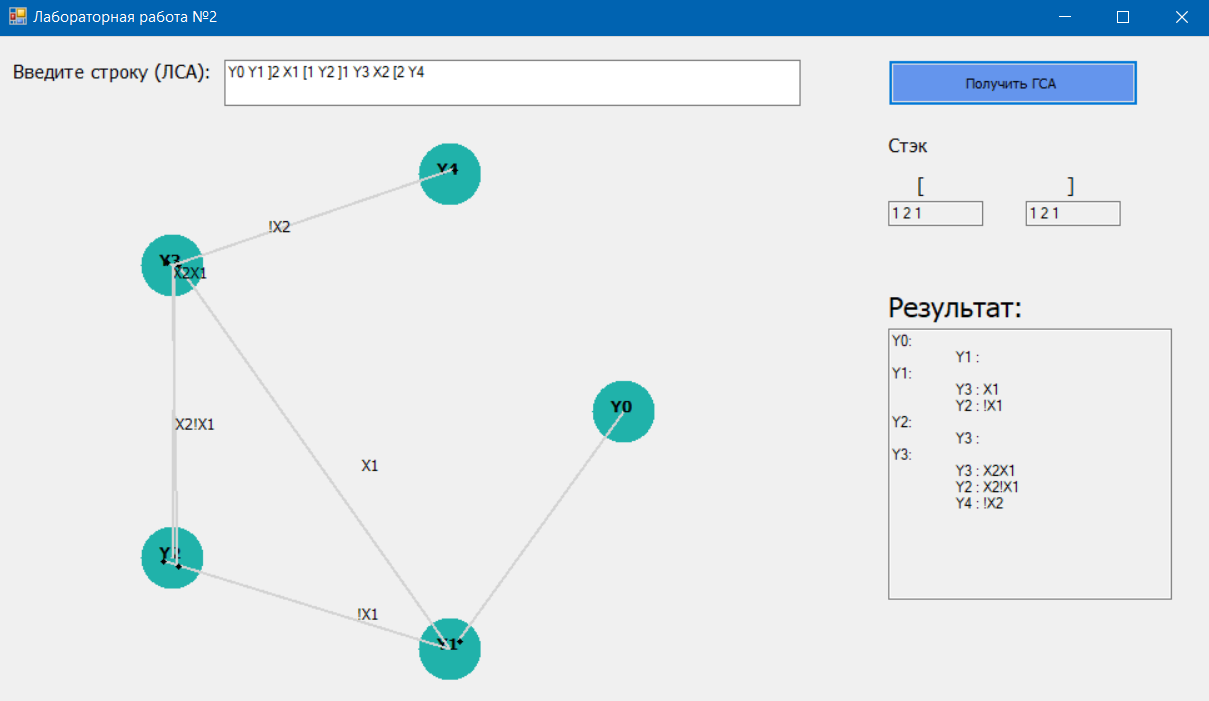
Принцип действия визуализации автомата заключается в том, что в первую очередь, происходит рисование всех вершин графа, которые соответствуют состояниям автомата, затем выполняется обход всех состояний. В матрице переходов определяется наличие перехода из текущего состояния в какое-либо другое, если переход существует, то происходит рисование ребра из вершины соответствующей текущему состоянию в новую вершину.

# Тестирование

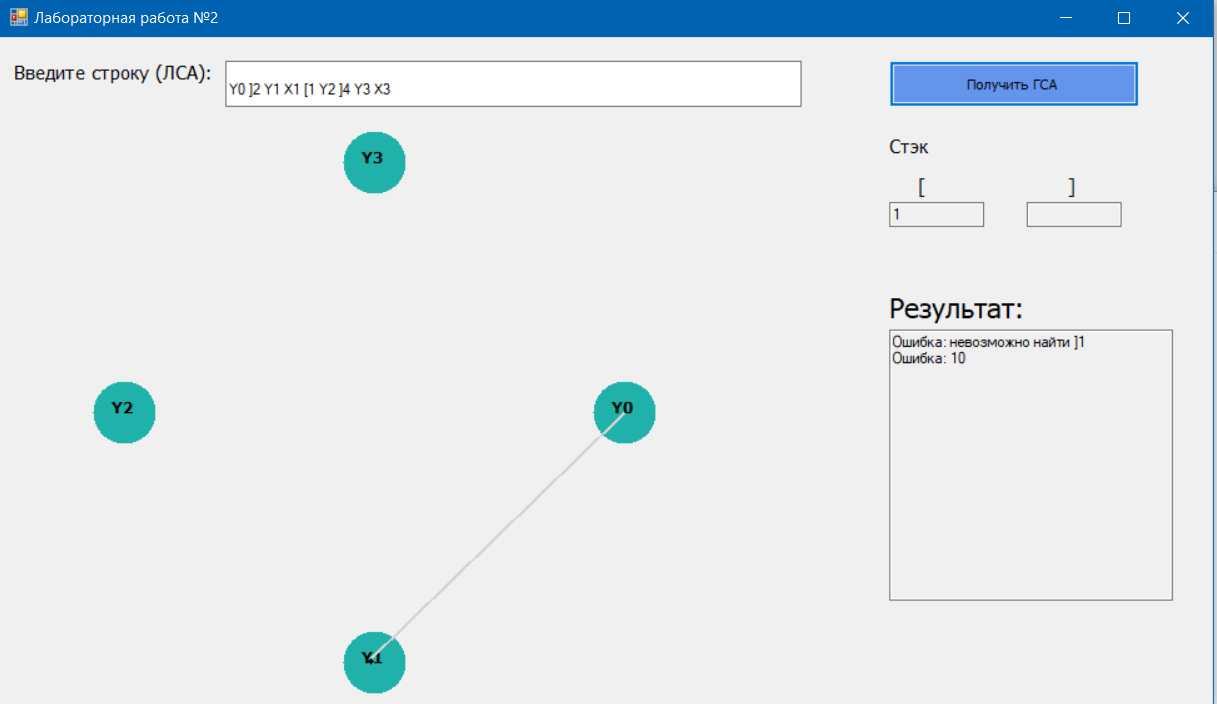
1. Входные данные - Y0 ]4 Y1 X1 [1 X2 [2 Y5 W [3 ]1 Y2 ]2 Y3 X3 [4 Y4 ]3 Y6



1. Входные данные – Y0 Y1 ]2 X1 [1 Y2 ]1 Y3 X2 [2 Y4



1. Входные данные – Y0 ]2 Y1 X1 [1 Y2 ]4 Y3 X3



В результате тестирования ошибки в работе программы обнаружены не были.

# Заключение

В процессе написания курсового проекта были:

* улучшены навыки в работе с микропрограммными автоматами;
* улучшены навыки в работе с графами;
* получены новые знания об автоматах с магазинной памятью;
* улучшены навыки в реализации алгоритмов;
* улучшено алгоритмическое мышление;
* улучшены навыки в программировании на языке С#;

В результате выполнения курсового проекта было разработано приложение по обработке частичного микропрограммного автомата. Данный инструмент полностью удовлетворяет требования поставленной задачи.

Разработанное приложение по обработке частичного микропрограммного автомата, может быть использовано в качестве готовой утилиты или в качестве одного из модулей сторонних приложений.

# Список литературы

1. Норенков И.П. Системы автоматизированного проектирования. Кн.1: Учеб. пособие для втузов. - М.: Высшая школа, 1986.
2. Способы описания работы дискретных устройств. Логические схемы алгоритмов. [Электронный ресурс] https://www.intuit.ru/studies/courses/1031/242/lecture/6230?page=3 (дата обращения: 07.01.2018)
3. В. А. Горбатов, А. В. Горбатов, М. В. Горбатова. Теория автоматов: учебник для студентов втузов. Высшая школа. Издательство «АСТ», 2008.
4. Белоусов А. И., Ткачев С. Б. Дискретная математика. — М.: МГТУ, 2006. — 743 с.

# Приложение. Исходный код

using System;

using System.Collections.Generic;

using System.Linq;

using System.Text;

using System.Threading.Tasks;

namespace apts\_lab1

{

// класс для реализации списка смежностей, хранит состоянии и условие перехода

class StateLSA

{

public string state;

public string cond\_if;

}

class Apts\_lab1

{

// Строки для вывода стэка скобок

public string stckLeftStr = "";

public string stckRightStr = "";

// Входная строка (ЛСА)

public string input;

// Список смежности

public List<List<StateLSA>> graph = new List<List<StateLSA>>();

// Строка для вывод результата

public string output;

// Счетчик цикла

public int cycle = 0;

// инициализация стэка для скобок

public Stack<int> stackLeft = new Stack<int>();

public Stack<int> stackRight = new Stack<int>();

// Функция проверки стэка на скобки

bool checkStack()

{

int count = Math.Max(stackLeft.Count, stackRight.Count);

// Пока стэк не пуст

while (stackLeft.Count > 0)

{

//Извлекаем левые полускобки

int arrowUp = stackLeft.Pop();

stckLeftStr += " " + arrowUp;

int arrowDown = -1;

//Извлекаем правые полускобки пока они не кончились

if (stackRight.Count != 0)

{

arrowDown = stackRight.Pop();

stckRightStr += " " + arrowDown;

}

// Если неравное кол-во скобок выводим ошибку

if (arrowDown != arrowUp)

{

output += string.Format("Ошибка: невозможно найти ']' {0}", arrowUp);

return false;

}

}

return true;

}

// инициализация входной строки

public Apts\_lab1(string \_input)

{

input = \_input;

}

// Функция добавления условия перехода

public void addCond\_if(List<StateLSA> list, string cond\_if)

{

foreach (StateLSA item in list)

item.cond\_if = cond\_if + item.cond\_if;

}

// Функция поиска правой полускобки

public int findRight(int number)

{

int index = input.IndexOf("]" + number);

if (index < 0)

{

output += "Ошибка: невозможно найти ]" + number + "\n";

throw new Exception();

}

return index;

}

// Поиск макс. состояния из Y

public int maxState()

{

int maxItem = 0;

for (int i = 0; i < input.Length; i++)

{

if (input[i] == 'Y')

{

i++;

int num = getNumber(ref i);

if (num > maxItem)

maxItem = num;

}

}

return maxItem;

}

// Функция получения номеров в строке

public int getNumber(ref int index)

{

int number = 0;

try

{

int space\_ind = input.IndexOf(' ', index);

if (space\_ind < 0)

space\_ind = input.Length;

string sub\_str = input.Substring(index, space\_ind - index);

number = int.Parse(sub\_str);

index = space\_ind + 1;

}

catch (Exception)

{

output += "Ошибка: " + index + "\n";

throw new Exception();

}

return number;

}

// Функция, реализующая получения состояния

public List<StateLSA> getState(int index)

{

if (cycle > 100)

{

output += "Ошибка цикла: " + index + "\n";

throw new Exception();

}

int x\_number = 0;

string cond\_if = "";

cycle++;

// Создаем список всех состояний

List<StateLSA> states = new List<StateLSA>();

while (index < input.Length)

{

// Если встречааем Y то создаем его состояние и получаем его номер

if (input[index] == 'Y')

{

index++;

int state\_number = getNumber(ref index);

states.Add(new StateLSA { state = state\_number.ToString(), cond\_if = "" });

cycle--;

return states;

}

// Если встречааем Х

else if (input[index] == 'X')

{

// Получаем его индекс и номер в строке

index++;

x\_number = getNumber(ref index);

cond\_if = "X" + x\_number;

if (input[index] == '[') // Стрелка вверх

{

// если условие перехода выполняется то (true)

index++;

int arrow\_number = getNumber(ref index);

int new\_ind = findRight(arrow\_number);

List<StateLSA> true\_states = getState(new\_ind);

addCond\_if(true\_states, cond\_if);

states.AddRange(true\_states);

// если условие перехода не выполняется (false)

List<StateLSA> false\_states = getState(index);

addCond\_if(false\_states, "!" + cond\_if);

states.AddRange(false\_states);

cycle--;

// Возвращаем состояние

return states;

}

}

// Если встречаем W

else if (input[index] == 'W')

{

index += 2;

cond\_if = "W";

// Ищем левые полускобки

if (input[index] == '[')

{

index++;

int arrow\_number = getNumber(ref index);

int new\_ind = findRight(arrow\_number);

List<StateLSA> true\_states = getState(new\_ind);

addCond\_if(true\_states, cond\_if);

states.AddRange(true\_states);

cycle--;

return states;

}

}

// Если пробел, то ничего не делаем и просто увел. индекс

else if (input[index] == ' ')

{

index++;

}

// Если ], то ищем индекс ]

else if (input[index] == ']')

{

index = input.IndexOf(' ', index);

}

else

{

// Вывод если ошибки

output += "Ошибка: " + index + "\n";

throw new Exception("Ошибка: " + index);

}

}

cycle--;

return null;

}

// Функция реализующая итоговый вывод всех состояний и переходов

public void printGraph()

{

for (int i = 0; i < graph.Count - 1; i++)

{

output += string.Format("Y{0}: ", i) + "\n";

for (int j = 0; j < graph[i].Count; j++)

{

output += string.Format("\t Y{0} : {1}", graph[i][j].state, graph[i][j].cond\_if) + "\n";

}

}

}

// Функция для разбития ХY во входной строке

public void findXY()

{

List<int> initX = new List<int>();

List<int> initY = new List<int>();

for (int i = 0; i < input.Length; i++)

{

// Если встречаем У то получаем номер, иначе выдаем ошибку

if (input[i] == 'Y')

{

i++;

int y\_num = getNumber(ref i);

if (initY.Contains(y\_num))

{

output += "Ошибка: " + i + "\n";

throw new Exception();

}

initY.Add(y\_num);

i--;

}

// Если встречаем Х то получаем номер, иначе выдаем ошибку

else if (input[i] == 'X')

{

i++;

int x\_num = getNumber(ref i);

if (initX.Contains(x\_num))

{

output += "Ошибка: " + i + "\n";

throw new Exception();

}

initX.Add(x\_num);

i--;

}

}

return;

}

// Функция получения самого ЛСА

public void getLSA()

{

int i = 0;

try

{

// Инициализация XY

findXY();

// Получаем все состояния

int max\_state = maxState();

// Создаем список смежности

graph = new List<List<StateLSA>>(new List<StateLSA>[max\_state + 1]);

for (i = 0; i < input.Length; i++)

{

if (input[i] == 'Y')

{

i++;

int state\_number = getNumber(ref i);

i--;

var state = getState(i);

graph[state\_number] = state;

}

}

printGraph();

}

catch (Exception e)

{

output += "Ошибка: " + i + "\n";

}

}

}

}

using System;

using System.Collections.Generic;

using System.ComponentModel;

using System.Data;

using System.Drawing;

using System.Linq;

using System.Text;

using System.Threading.Tasks;

using System.Windows.Forms;

using apts\_lab1;

namespace apts\_lab2

{

public partial class Form1 : Form

{

// Список смежности

List<List<apts\_lab1.StateLSA>> graph;

// Список состояний

List<State> states = new List<State>();

// Список линий

List<Line> lines = new List<Line>();

Random rand = new Random();

// Инициализация формы

public Form1()

{

InitializeComponent();

}

//Отрисовка состояний и линий

protected override void OnPaint(PaintEventArgs e)

{

base.OnPaint(e);

Graphics g = e.Graphics;

for (int i = 0; i < states.Count; i++)

states[i].draw(g);

for (int i = 0; i < lines.Count; i++)

lines[i].draw(g, rand);

}

// Получения состояния У

void getState()

{

int R = 200;

int centerX = 300;

int centerY = 300;

double alp = 2 \* Math.PI / states.Count;

for(int i = 0; i < states.Count; i++)

{

states[i].x = centerX + (int)(R \* Math.Cos(i \* alp));

states[i].y = centerY + (int)(R \* Math.Sin(i \* alp));

}

}

// Получение ГСА

void getGSA()

{

//Состояние

for(int i = 0; i < graph.Count; i++)

{

states.Add(new State(0, 0, "Y" + i));

}

getState();

// Линии

for(int i = 0; i < graph.Count; i++)

{

if (graph[i] == null)

continue;

State s1 = states[i];

for(int j = 0; j < graph[i].Count; j++)

{

State s2 = State.getState(states, graph[i][j]);

lines.Add(new Line(s1, s2, graph[i][j].cond\_if));

}

}

}

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

{

states.Clear();

lines.Clear();

string input = textBox1.Text;

Apts\_lab1 lab1 = new Apts\_lab1(input);

lab1.getLSA();

textBox2.Text = lab1.output.Replace("\n", "\r\n");

if (lab1.stackLeft.Count > 0)

{

textBox3.Text = string.Join(" ", lab1.stackLeft);

textBox4.Text = string.Join(" ", lab1.stackRight);

}

else

{

textBox3.Text = lab1.stckLeftStr;

textBox4.Text = lab1.stckRightStr;

}

graph = lab1.graph;

getGSA();

Refresh();

}

}

// Класс для отрисовки состояния

class State

{

Pen p;

SolidBrush fp = new SolidBrush(Color.LightSeaGreen);

SolidBrush p\_str = new SolidBrush(Color.Black);

public int x, y;

public int r = 50;

string text;

Font font = new Font("Tahoma", 10, FontStyle.Bold);

public State(int \_x = 0, int \_y = 0, string \_text = "Y")

{

x = \_x;

y = \_y;

text = \_text;

p = new Pen(Color.White);

}

// Отрисовка круга

public void draw(Graphics g)

{

int rectX = x - r / 2;

int rectY = y - r / 2;

g.FillEllipse(fp, rectX, rectY, r, r);

g.DrawString(text, font, p\_str, x - r / 4, y - r / 4);

}

// Отрисовка точек к линиям

public Point randomPoint(Random rand)

{

int range = r / 6;

int randX = rand.Next(x - range, x + range);

int randY = rand.Next(y - range, y + range);

return new Point(randX, randY);

}

// Получение состояния из списка смежностей

public static State getState(List<State> states, apts\_lab1.StateLSA state)

{

int stateNum = int.Parse(state.state);

return states[stateNum];

}

}

// Класс для отрисовки линий

class Line

{

Pen p;

State s1;

State s2;

string cond\_if;

Font font = new Font("Tahoma", 9, FontStyle.Regular);

// Инициализация

public Line(State \_s1, State \_s2, string \_cond\_if)

{

s1 = \_s1;

s2 = \_s2;

cond\_if = \_cond\_if;

p = new Pen(Color.LightGray, 2.0f);

}

// Отрисовка линии

public void draw(Graphics g, Random rand)

{

int cursorSize = 2;

Point s2Point = s2.randomPoint(rand);

g.DrawLine(p, s1.x, s1.y, s2Point.X, s2Point.Y);

g.DrawEllipse(new Pen(Color.Black, 2), s2Point.X - cursorSize/4, s2Point.Y - cursorSize/4, cursorSize, cursorSize);

Point cond\_ifPoint = new Point(s1.x + (s2Point.X - s1.x) / 3, s1.y + (s2Point.Y - s1.y) / 2);

g.DrawString(cond\_if, font, Brushes.Black, cond\_ifPoint);

}

}

}