Министерство науки и высшего образования Российской Федерации
Федеральное государственное бюджетное образовательное учреждение
высшего профессионального образования
«Санкт-Петербургский государственный технологический институт
(технический университет)»

Факультет: Информационных технологий и управления

Кафедра: Систем автоматизированного проектирования и управления

Уровень подготовки: Бакалавр

Учебная дисциплина: Математическая логика и теория алгоритмов

П Р А К Т И Ч Е С К А Я Р А Б О Т А

тема: РАЗРАБОТКА ЭМУЛЯТОРА МАШИНЫ ТЬЮРИНГА

Преподаватель _____ Плонский В.Ю.

Студент _____ Гусев А.А. 494 группа

Санкт-Петербург
2021

Постановка задачи

Необходимо разработать эмулятор машины Тьюринга, отвечающий следующим требованиям:

- Возможность задавать состояния машины;
- Возможность ввода пользовательской ленты;
- Возможность ввода пользовательской программы;
- Возможность выполнение введенной программы по шагам, используя исходные данные.

Исходные данные

В качестве исходных данных программа использует пользовательский ввод значений в специальные поля для ввода. Доступно указание количества необходимые пустых ячеек на ленте, пользовательского алфавита, ленты и таблицы состояний.

Особые ситуации

Необходимо рассмотреть следующие особые ситуации:

- Указатель на ячейку ленты не должен выйти за пределы ленты;
- Повторение элементов алфавита;
- Наличие на ленте символов, необъявленных в алфавите;
- Отсутствие на ленте необходимого (указанного изначально) количества пустых ячеек;
- Ввод пустой ленты;
- Наличие в команде программе указателя на несуществующую ячейку состояния машины;
- Наличие в команде программе несуществующего указателя сдвига указателя по ленте;
- Наличие в команде программе символа, предназначенного для замены элемента на ленте, который не был объявлен в алфавите;
- Отсутствие или неполный текст команды в ячейке таблицы состояний, на которую указала предыдущая команда.

Математические методы и алгоритмы решения задач

Поставленная задача не требует использования особых математических методов.

Структура программы

Программа разбита на 5 классов.

Основная последовательность работы программы – ожидания решения пользователя. Программа ожидает пользовательские нажатия на доступные в тот или иной момент элементы управления. После ввода корректный данных пользователь может выполнить программу пошагово или в автоматическом режиме, указав необходимую задержку между шагами. В

процессе работы программа выделяет цветом команду текущего шага и меняет содержимое ленты. Кнопка «Настройка» позволяет включить или отключить отображение справки перед запуском основной программы. Кнопка «Справка» открывает информацию о программе. Эмулятор продолжает свою работу до тех пор, пока его не закроет пользователь в правом верхнем углу или с помощью средств операционной системы.

Блок-схема алгоритма

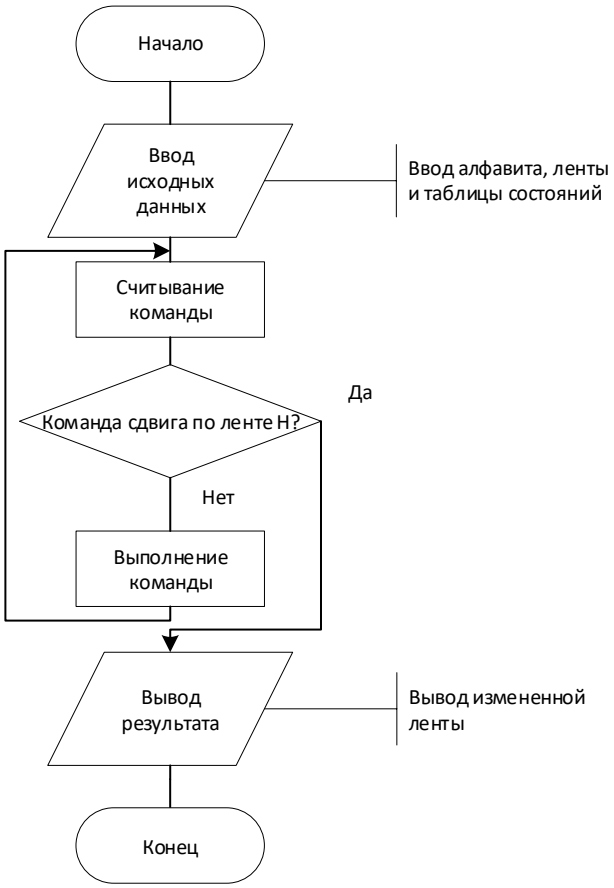

Рисунок 1 - Блок-схема алгоритма программы

Описание хода выполнения практической работы

В ходе практический работы было создано решение (Solution) интегрированной среде разработки Microsoft Visual Studio C# 2019. В нём был создан проект.

После написания интерфейса и обработки пользовательского взаимодействия с ним, был разработан основного цикла работы программы. Затем были обработаны различные исключения возникающие в процессе работы эмулятора, произведена отладка и тестирование

Результаты работы программы

Перед запуском эмулятора необходимо указать необходимого количества пустых ячеек на ленте:

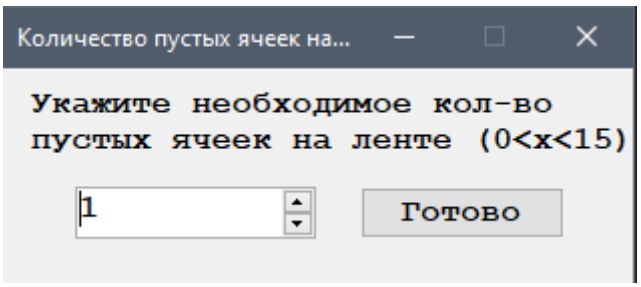

Рисунок 2 - Ввод необходимого количества пустых ячеек

После указания значения открывается основное окно эмулятора, где расположены поля для ввода алфавита и ленты, а также все основные элементы управления эмулятором:

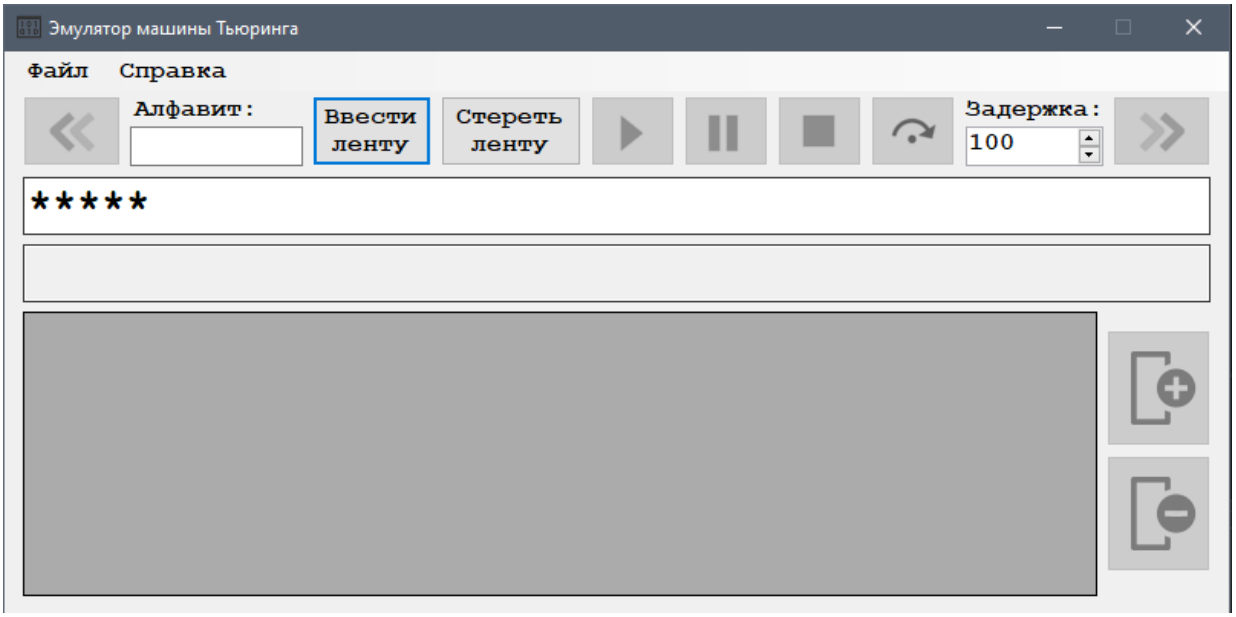

Рисунок 3 - Основное окно эмулятора

При вызове меню «Файл» пользователь может воспользоваться дополнительными способами взаимодействия с программным комплексом:

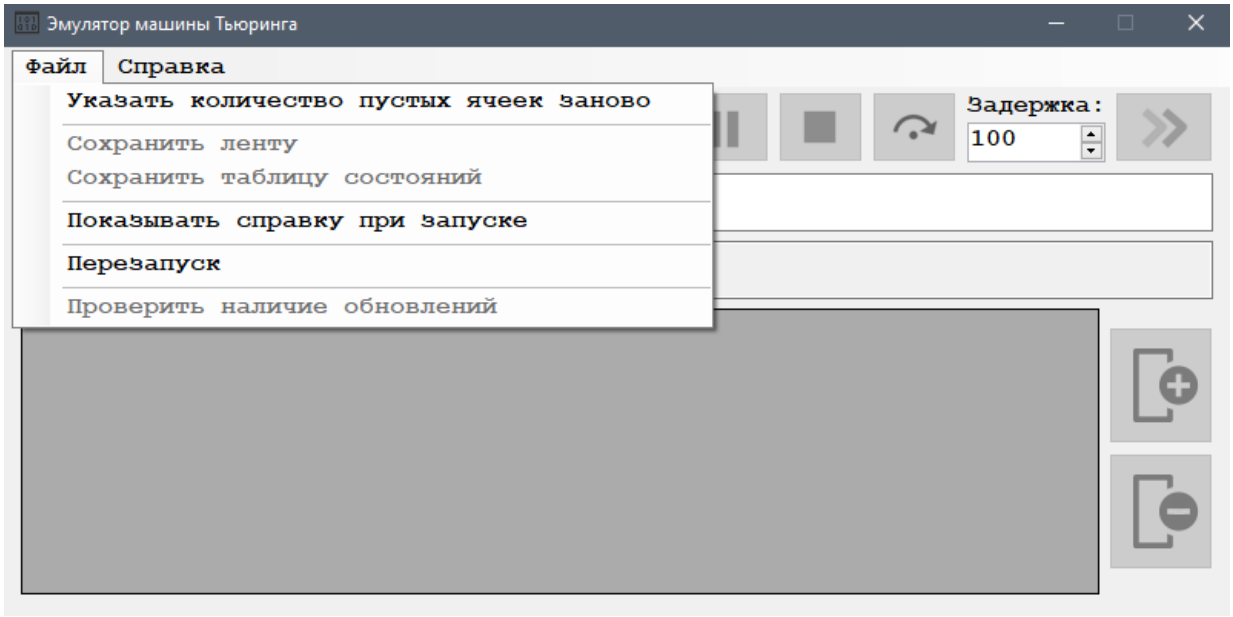

Рисунок 4 - Меню «Файл»

Меню «Справка» показывает информацию об авторе, программе и инструкцию как пользоваться эмулятором:

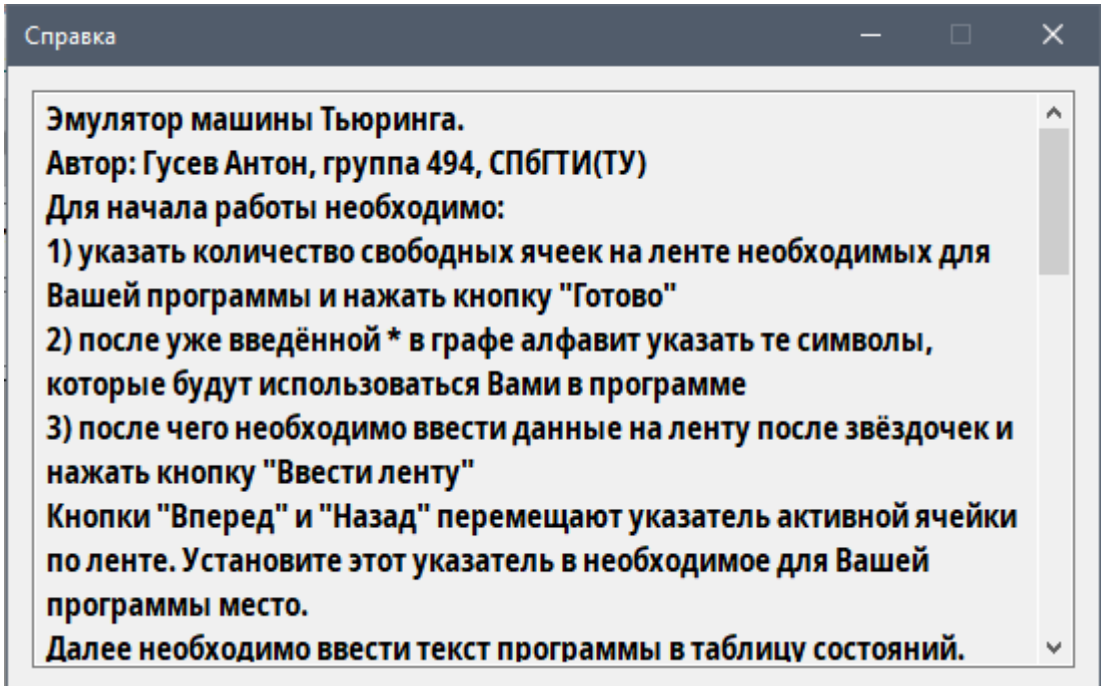

Рисунок 5 - Меню справка

После корректного ввода алфавита и ленты и нажатия кнопки «Ввести ленту» эмулятор достроит ленту пустыми ячейками, выведет полосу указателя на экран, а также сформирует таблицу состояний для ввода пользовательской программы:

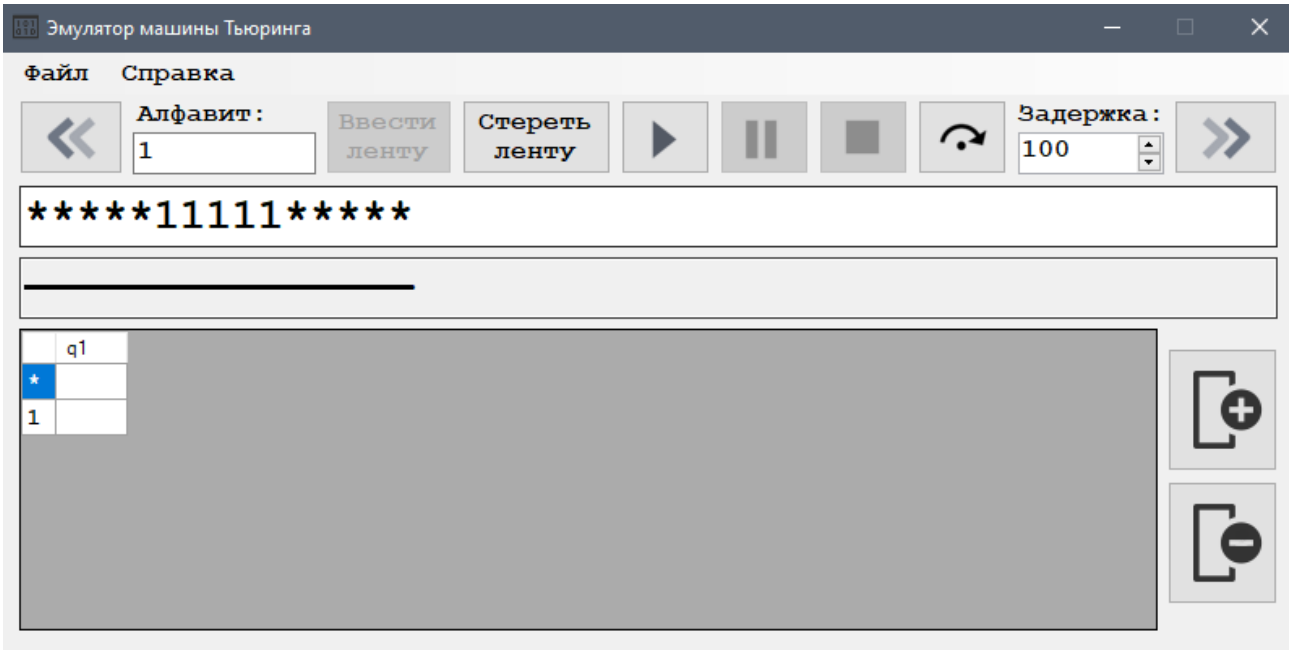

Рисунок 6 - Пример работы программы

Пользователь может добавлять и удалять столбцы сформированной таблицы состояний.

После ввода программы пользователь может выполнить ее пошагово или в автоматическом режиме, указав задержку:

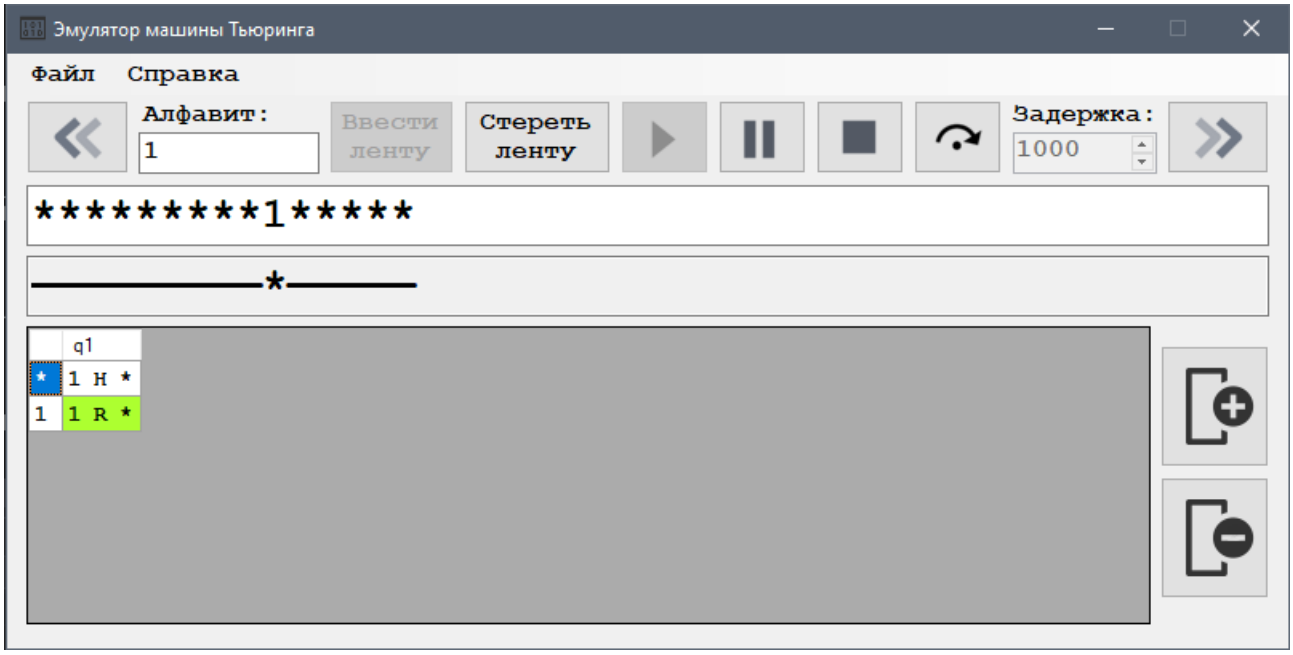

Рисунок 7 - Программа, стирающая единицы с ленты

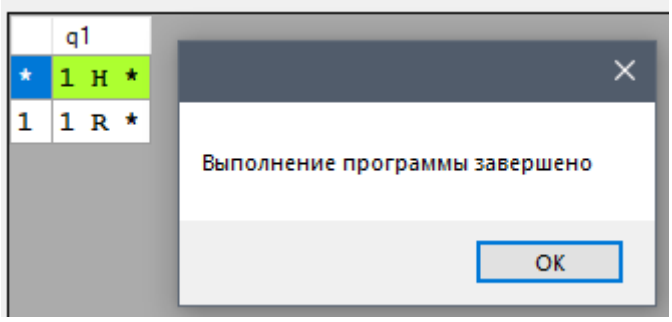

Рисунок 8 - Оповещение о завершении работы программы

Исходный текст программы

[Начало программы ---]
[Начало Program.cs ---]

```
using System;
using System.Windows.Forms;

namespace TuringMachine
{
    static class Program
    {
        [STAThread]
        static void Main()
        {
            Application.EnableVisualStyles();
            Application.SetCompatibleTextRenderingDefault(false);
            Application.Run(new MainForm());
        }
    }
}
```

[Конец Program.cs ---]
[Начало Empty.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 TuringMachine
{
    public partial class Empty : Form
    {
        ProcessWorkingMachine work = new ProcessWorkingMachine();
        public Empty(ProcessWorkingMachine w)
        {
            InitializeComponent();
            work = w;
            MaximizeBox = false;
        }

        private void Button1_Click(object sender, EventArgs e)
        {
            work.CountEmpty = Convert.ToInt32(numericUpDown1.Value);
            Close();
        }
    }
}
```

[Конец Empty.cs ---]

[Начало InfoForm.cs ---]

```
using System.Windows.Forms;

namespace TuringMachine
{
    public partial class InfoForm : Form
    {
        public InfoForm()
        {
            InitializeComponent();
            MaximizeBox = false;
        }

    }
}
```

[Конец InfoForm.cs ---]

[Начало MainWindow.cs ---]

```
using System;
using System.Collections.Generic;
using System.Drawing;
using System.Linq;
using System.Reflection;
using System.Threading.Tasks;
using System.Windows.Forms;
using System.Xml;

namespace TuringMachine
{

    public partial class MainForm : Form
    {
        public static ProcessWorkingMachine work = new ProcessWorkingMachine();
        private List<string> pointers = new List<string>();
        private List<string> LineList = new List<string>();
        private List<string> Alph = new List<string>();
        private int pointerPosition = work.CountEmpty - 1;

        public MainForm()
        {
            InitializeComponent();
            MaximizeBox = false;
            saveFileDialog.Filter = "Text (*.txt)|*.txt";

            if (Settings.Default.Show == true)
            {
                InfoToolStripMenuItem_Click(null, null);
                ShowInfoOnStartToolStripMenuItem.Checked = true;
            }
            else ShowInfoOnStartToolStripMenuItem.Checked = false;

            var f = new Empty(work);
            f.ShowDialog();
```

```
            if (work.CountEmpty == 0) work.CountEmpty = 1;
            for (int i = 0; i < work.CountEmpty; i++)
            {
                textBoxLine.Text += "*";
            }
        }

        public bool Check(string text) //проверка наличия пустых ячеек
        {
            for (int i = 0; i < work.CountEmpty; i++)
            {
                if (text[i] != '*')
                {
                    return false;
                }
            }
            return true;
        }
        public void Erase()
        {
            ButtonBack.Enabled = false;
            ButtonFront.Enabled = false;
            SaveLineToolStripMenuItem.Enabled = false;
            SaveTableToolStripMenuItem.Enabled = false;

            textBoxPointer.Text = "";
            textBoxLine.Text = "";
            for (int i = 0; i < work.CountEmpty; i++)
            {
                textBoxLine.Text += "*";
            }
            Alph.Clear();
            LineList.Clear();
        }
        private void CreatingLinePointer() // создает полосу указателя
        {
            textBoxPointer.Text = "";
            pointers.Clear();
            for (int i = 0; i < textBoxLine.TextLength; i++)
            {
                pointers.Add("—");
            }
        }
        private void RecoverPointer(bool left)//восстанавливает линию указателя после пойманного
исключения
        {
            if (left)
            {
                pointerPosition++;
                pointers[pointerPosition] = "*";
                for (int i = 0; i < textBoxLine.TextLength; i++)
                {
                    textBoxPointer.Text += pointers[i];
                }
                return;
            }
            else
            {
                pointerPosition--;
                pointers[pointerPosition] = "*";
                for (int i = 0; i < textBoxLine.TextLength; i++)
                {
                    textBoxPointer.Text += pointers[i];
                }
                return;
            }
        }

        private void ButtonEnterLine_Click(object sender, EventArgs e) // создание ленты
        {
            Alph.Clear();
            int counter = 0;
            for (int i = 0; i < work.CountEmpty; i++) // добавление пустого пространства в конец
ленты
            {
                textBoxLine.Text += "*";
            }
            for (int i = 0; i < textBoxLine.TextLength; i++) // формирование листа ленты
```

```
            {
                LineList.Add(textBoxLine.Text[i].ToString());
            }

            Alph.Add("*");
            for (int i = 0; i < textBoxAlph.TextLength; i++) // алфавит лист
            {
                Alph.Add(textBoxAlph.Text[i].ToString());
            }

            for (int i = 0; i < textBoxLine.TextLength; i++) // проверка на вхождение символов ленты
в алфавит
            {
                if (!Alph.Contains(LineList[i]))
                {
                    counter++;
                }
            }

            if (counter == 0 && (textBoxLine.TextLength != work.CountEmpty * 2) &&
Alph.Distinct().ToList().Count == Alph.Count && Check(textBoxLine.Text))
            { // если лента не содержит иных символов, лента не пуста, алфавит не содержит повторов,
лента содержит установленные пустые ячейки, то
                pointerPosition = work.CountEmpty - 1; // позиция указателя на линии слева он
первого
                CreatingLinePointer(); // создение полосы указателя
                for (int i = 0; i < textBoxLine.TextLength; i++)
                {
                    textBoxPointer.Text += pointers[i]; //вывод полосы указателя
                }
                ButtonBack.Enabled = true; // разблок кнопок управления
                ButtonFront.Enabled = true;

                if (dataGridView1.RowCount == 0) CreateTable(); // если таблица не создана ранее, то
создаём
                else if (MessageBox.Show("Стереть таблицу?", "Подтверждение",
MessageBoxButtons.YesNo) == DialogResult.Yes) CreateTable(); // пересоздаем по желанию

                ButtonEnterLine.Enabled = false; // блок кнопки создать

                ButtonAddColumns.Enabled = true; // разблок управления таблицей
                ButtonDeleteColumns.Enabled = true;

                SaveLineToolStripMenuItem.Enabled = true;
                SaveTableToolStripMenuItem.Enabled = true;

                ButtonStep.Enabled = true;
                ButtonStart.Enabled = true;

            }
            else if (counter != 0)
            {
                MessageBox.Show("Лента содержит символы, необъявленные в алфавите. Проверьте
правильность ленты и уточните алфавит.",
                    "Ошибка", MessageBoxButtons.OK, MessageBoxIcon.Error);
                Erase();
            }
            else if (textBoxLine.TextLength == work.CountEmpty * 2)
            {
                MessageBox.Show("Вы ввели пустую ленту. Укажите хотя бы одно значение",
                    "Ошибка", MessageBoxButtons.OK, MessageBoxIcon.Error);
                Erase();
            }
            else if (Alph.Distinct().ToList().Count != Alph.Count)
            {
                MessageBox.Show("Алфавит содержит повторяющиеся элементы. Их необходимо удалить.",
                   "Ошибка", MessageBoxButtons.OK, MessageBoxIcon.Error);
                Erase();
            }
            else if (!Check(textBoxLine.Text))
            {
                MessageBox.Show("Для корректной работы в начале ленты необходимо столько символов
\"*\" (звёздочка), сколько вы указали изначально!",
                   "Ошибка", MessageBoxButtons.OK, MessageBoxIcon.Error);
                Erase();
            }
            counter = 0;
        }
```

```
        private void ButtonBack_Click(object sender, EventArgs e) // перемещение указателя по полосе
влево (назад)
        {
            try
            {
                CreatingLinePointer();// пересоздание полосы указателя
                pointerPosition--; // сдвиг влево
                pointers[pointerPosition] = "*"; // перемещение указателя
                for (int i = 0; i < textBoxLine.TextLength; i++)
                {
                    textBoxPointer.Text += pointers[i]; // вывод в текстбокс
                }
            }
            catch (ArgumentOutOfRangeException)
            {
                MessageBox.Show("Указатель находится в крайнем левом положении.\nСдвиг левее
невозможен", "Ошибка",
                   MessageBoxButtons.OK,
                   MessageBoxIcon.Error);
                RecoverPointer(true);

            }
        }

        private void ButtonFront_Click(object sender, EventArgs e)// перемещение указателя по полосе
вправо (вперед)
        {
            try
            {
                CreatingLinePointer();// пересоздание полосы указателя
                pointerPosition++;// сдвиг влево
                pointers[pointerPosition] = "*"; // перемещение указателя
                for (int i = 0; i < textBoxLine.TextLength; i++)
                {
                    textBoxPointer.Text += pointers[i];// вывод в текстбокс
                }
            }
            catch (ArgumentOutOfRangeException)
            {
                MessageBox.Show("Указатель находится в крайнем правом положении.\nСдвиг правее
невозможен", "Ошибка",
                    MessageBoxButtons.OK,
                    MessageBoxIcon.Error);
                RecoverPointer(false);
            }
        }

        public void CreateTable()// создание таблицы состояний
        {
            dataGridView1.Columns.Clear();
            dataGridView1.Columns.Add("zero", ""); dataGridView1.AutoSizeColumnsMode =
DataGridViewAutoSizeColumnsMode.AllCells;
            dataGridView1.Columns.Add("1", "q1");
            for (int i = 0; i < Alph.Count; i++)
            {
                dataGridView1.Rows.Add(Alph[i]);
            }
            dataGridView1.Columns[0].ReadOnly = true;
            dataGridView1.AllowUserToResizeColumns = false;
            dataGridView1.AllowUserToResizeRows = false;
            dataGridView1.AllowUserToAddRows = false;
            dataGridView1.DefaultCellStyle.Font = new Font("Courier New", 10, FontStyle.Bold);
        }

        private void buttonEraseLine_Click(object sender, EventArgs e) // стереть ленту
        {
            ButtonEnterLine.Enabled = true; // разблок кнопки ввести ленту
            ButtonStep.Enabled = false; // блок управляющих кнопок
            ButtonStart.Enabled = false;

            textBoxLine.Text = ""; // очиска ленты
            for (int i = 0; i < work.CountEmpty; i++)
            {
                textBoxLine.Text += "*";// добавление пустых ячеек (*)
            }
            textBoxPointer.Text = "";// очистка полосы указателя
            pointerPosition = work.CountEmpty - 1; //позиция указателя слева от первого элемента
ленты
            LineList.Clear();// очистка листа ленты
```

```
            work.Command = null; work.Direction = null;
            work.NextColumn = null; work.ReplaceOnIt = null;
        }

        private void ResetToolStripMenuItem_Click(object sender, EventArgs e)
        {
            Application.Restart(); // рестарт приложения
        }

        private void InfoToolStripMenuItem_Click(object sender, EventArgs e)
        {
            var info = new InfoForm();
            info.ShowDialog();// инфо
        }

        public void UpdateLine() // апдейт ленты
        {
            textBoxLine.Text = ""; // стираем
            for (int i = 0; i < LineList.Count; i++)
            {
                textBoxLine.Text += LineList[i]; // выводим новую в соответствии с LineList
            }
        }
        private void ButtonStep_Click(object sender, EventArgs e)
        {
            foreach (DataGridViewRow row in dataGridView1.Rows)
            {
                foreach (DataGridViewCell cell in row.Cells)
                {
                    cell.Style.BackColor = Color.White; // красив в белый
                }
            }

            if (work.NextColumn == null)
            {
                work.NextColumn = "1"; // если это первый шаг то начинаем с q1
            }

            work.CurrentContentCell = LineList[pointerPosition]; //текущая клетка - значение ленты
где стоит указатель снизу


dataGridView1.Rows[Alph.IndexOf(work.CurrentContentCell)].Cells[int.Parse(work.NextColumn)].Style.Ba
ckColor = Color.GreenYellow; // красим в цвет, что мы там находимся

            try
            {
                work.CurrentContentCell = LineList[pointerPosition]; //текущая клетка - значение
ленты где стоит указатель снизу
                work.Command = (string)dataGridView1[work.NextColumn,
Alph.IndexOf(work.CurrentContentCell)].Value; // берем команду из ячейки таблицы
                work.SplittedCommand = work.Command.Split(' ').ToList(); // сплитим ее по пробелам

                if (dataGridView1.Columns.Contains(work.SplittedCommand[0]))
                {
                    work.NextColumn = work.SplittedCommand[0]; // 0 элемент это след колонка
                }
                else
                {
                    throw new NextColumnException();
                }

                work.Direction = work.SplittedCommand[1]; // 1 элемент это направление L R или H

                if (Alph.Contains(work.SplittedCommand[2]))
                {
                    work.ReplaceOnIt = work.SplittedCommand[2]; // 2 элемент это то на что заменяем
                }
                else
                {
                    throw new AlphabetException();
                }

                LineList[pointerPosition] = work.ReplaceOnIt; // обновляем элемент в листе
                UpdateLine(); // обновляем на ленте
                if (work.Direction == "R") ButtonFront_Click(null, null); // сдвиг вправо
                else if (work.Direction == "L") ButtonBack_Click(null, null); // сдвиг влево
                else if (work.Direction == "H")// стоп машина
                {
```

```
                    MessageBox.Show("Выполнение программы завершено");
                    ButtonStart.Enabled = true;
                    ButtonStop_Click(null, null);
                }
                else
                {
                    throw new WrongDirectionException();
                }
            }
            catch (NullReferenceException)
            {
                string textError = "Ячейка (q" + work.NextColumn + ";" + LineList[pointerPosition] +
") не содержит команды";

dataGridView1.Rows[Alph.IndexOf(work.CurrentContentCell)].Cells[int.Parse(work.NextColumn)].Style.Ba
ckColor = Color.Red;
                MessageBox.Show(textError, "Ошибка", MessageBoxButtons.OK, MessageBoxIcon.Error);
                ButtonStart.Enabled = true;
            }
            catch (AlphabetException)
            {
                string textError = "Ячейка (q" + work.NextColumn + ";" + LineList[pointerPosition] +
") содержит необъявленный в алфавите символ. Замена на ленте невозможна.";

dataGridView1.Rows[Alph.IndexOf(work.CurrentContentCell)].Cells[int.Parse(work.NextColumn)].Style.Ba
ckColor = Color.Red;
                MessageBox.Show(textError, "Ошибка", MessageBoxButtons.OK, MessageBoxIcon.Error);
                ButtonStart.Enabled = true;
            }
            catch (WrongDirectionException)
            {
                string textError = "Ячейка (q" + work.NextColumn + ";" + LineList[pointerPosition] +
") содержит неверное значение направления сдвига по ленте. Сдвиг по ленте невозможен.";

dataGridView1.Rows[Alph.IndexOf(work.CurrentContentCell)].Cells[int.Parse(work.NextColumn)].Style.Ba
ckColor = Color.Red;
                MessageBox.Show(textError, "Ошибка", MessageBoxButtons.OK, MessageBoxIcon.Error);
                ButtonStart.Enabled = true;
            }
            catch (NextColumnException)
            {
                string textError = "Ячейка (q" + work.NextColumn + ";" + LineList[pointerPosition] +
") содержит номер несуществующего состояния. Переход в указанное состояние невозможен";

dataGridView1.Rows[Alph.IndexOf(work.CurrentContentCell)].Cells[int.Parse(work.NextColumn)].Style.Ba
ckColor = Color.Red;
                MessageBox.Show(textError, "Ошибка", MessageBoxButtons.OK, MessageBoxIcon.Error);
                ButtonStart.Enabled = true;
            }

        }

        private void ButtonAddColumns_Click(object sender, EventArgs e) // добавление столбца с
уникальным номером
        {
            string nameNewColumn = dataGridView1.Columns[dataGridView1.Columns.Count - 1].Name;
            dataGridView1.Columns.Add((int.Parse(nameNewColumn) + 1).ToString(), "q" +
(int.Parse(nameNewColumn) + 1).ToString());
        }

        private void ButtonDeleteColumns_Click(object sender, EventArgs e) // удаление столбца кроме
нулевого и q1
        {
            if (dataGridView1.CurrentCell.ColumnIndex > 1)
                dataGridView1.Columns.RemoveAt(dataGridView1.CurrentCell.ColumnIndex);
        }

        private void ShowInfoOnStartToolStripMenuItem_Click(object sender, EventArgs e)
        {
            if (ShowInfoOnStartToolStripMenuItem.Checked)
            {
                ShowInfoOnStartToolStripMenuItem.Checked = false;
                Settings.Default.Show = false;
                Settings.Default.Save();
            }
            else
            {
                ShowInfoOnStartToolStripMenuItem.Checked = true;
                Settings.Default.Show = true;
```

```
                Settings.Default.Save();

            }
        }

        private void SaveLineToolStripMenuItem_Click(object sender, EventArgs e)
        {
            if (saveFileDialog.ShowDialog() == DialogResult.Cancel) return;
            string fileOutputPath = saveFileDialog.FileName;
            saveFileDialog.FileName = string.Empty;
            System.IO.File.WriteAllText(fileOutputPath, "Лента: \n");
            for (int i = 0; i < LineList.Count; i++)
            {
                System.IO.File.AppendAllText(fileOutputPath, LineList[i]);
            }
        }

        private void SaveTableToolStripMenuItem_Click(object sender, EventArgs e)
        {
            Microsoft.Office.Interop.Excel.Application ExcelApp = new
Microsoft.Office.Interop.Excel.Application();
            Microsoft.Office.Interop.Excel.Workbook ExcelWorkBook;
            Microsoft.Office.Interop.Excel.Worksheet ExcelWorkSheet;
            //Книга
            ExcelWorkBook = ExcelApp.Workbooks.Add(Missing.Value);
            //Таблица
            ExcelWorkSheet =
(Microsoft.Office.Interop.Excel.Worksheet)ExcelWorkBook.Worksheets.get_Item(1);
            ExcelApp.Cells[1, 1] = "Таблица состояний:";

            for (int i = 0; i < dataGridView1.Columns.Count; i++)
            {
                ExcelApp.Cells[2, i + 1] = dataGridView1.Columns[i].HeaderText;
            }

            for (int i = 0; i < dataGridView1.Columns.Count; i++)
            {
                for (int j = 0; j < dataGridView1.RowCount; j++)
                {
                    ExcelApp.Cells[j + 3, i + 1] = dataGridView1[i, j].Value;
                }
            }
            ExcelApp.Cells.HorizontalAlignment =
Microsoft.Office.Interop.Excel.XlHAlign.xlHAlignRight;
            ExcelApp.Visible = true;
            ExcelApp.UserControl = true;
        }

        private async void ButtonStart_Click(object sender, EventArgs e)
        {
            ButtonStart.Enabled = false;
            DelayNumericUpDown.Enabled = false;

            ButtonPause.Enabled = true;
            ButtonStop.Enabled = true;

            int delay = int.Parse(DelayNumericUpDown.Text);
            while (!ButtonStart.Enabled)
            {
                ButtonStep_Click(null, null);
                await Task.Delay(delay);
            }
        }

        private void ButtonPause_Click(object sender, EventArgs e)
        {
            ButtonStart.Enabled = true;
            ButtonPause.Enabled = false;

        }

        private void ButtonStop_Click(object sender, EventArgs e)
        {
            foreach (DataGridViewRow row in dataGridView1.Rows)
            {
                foreach (DataGridViewCell cell in row.Cells)
                {
                    cell.Style.BackColor = Color.White; // красим в белый
                }
```

```
            }

            pointerPosition = work.CountEmpty - 1;

            work.Command = null; work.Direction = null;
            work.NextColumn = null; work.ReplaceOnIt = null;

            ButtonStart.Enabled = true;
            DelayNumericUpDown.Enabled = true;

            ButtonPause.Enabled = false;
            ButtonStop.Enabled = false;

        }

        private void EnterEmptyAgainToolStripMenuItem_Click(object sender, EventArgs e)
        {
            buttonEraseLine_Click(null, null);
            work.CountEmpty = 0;
            var f = new Empty(work);
            f.ShowDialog();

            if (work.CountEmpty == 0) work.CountEmpty = 1;
            for (int i = 0; i < work.CountEmpty; i++)
            {
                textBoxLine.Text += "*";
            }
        }

        // область описания пользовательских исключений
        public class AlphabetException : ApplicationException
        {
            public AlphabetException() : base() { }
        }
        public class WrongDirectionException : ApplicationException
        {
            public WrongDirectionException() : base() { }
        }
        public class NextColumnException : ApplicationException
        {
            public NextColumnException() : base() { }
        }
    }
}
```

[Конец MainWindow.cs ---]

[Начало ProcessWorkingMachine.cs ---]

```
using System.Collections.Generic;

namespace TuringMachine
{
    public class ProcessWorkingMachine
    {
        public int CountEmpty { get; set; }
        public string CurrentContentCell { get; set; }
        public string Command { get; set; }
        public List<string> SplittedCommand { get; set; }
        public string NextColumn { get; set; }
        public string Direction { get; set; }
        public string ReplaceOnIt { get; set; }

    }
}
```

[Конец ProcessWorkingMachine.cs ---]

[Конец программы ---]