МИНИСТЕРСТВО ОБРАЗОВАНИЯ РЕСПУБЛИКИ БЕЛАРУСЬ
Учреждение образования
«Витебский государственный технологический университет»

# РАЗРАБОТКА МУЛЬТИМЕДИЙНЫХ ПРИЛОЖЕНИЙ. МНОГООКОННЫЕ ПРИЛОЖЕНИЯ

Методические указания по выполнению
лабораторных работ для студентов специальностей
1-19 01 01-06 «Дизайн виртуальной среды» и
6-05-0211-05 «Графический дизайн и мультимедиадизайн»

Витебск
2025

УДК 004.9
ББК 32.97
Р17

Составители:

Е. Б. Дунина, А. С. Соколова, А. А. Корниенко

Одобрено кафедрой «Информационные системы и технологии»
УО «ВГТУ», протокол № 13 от 15.05.2025.

Рекомендовано к изданию редакционно-издательским
советом УО «ВГТУ», протокол № 9 от 28.05.2025.

**Разработка мультимедийных приложений. Многооконные приложения** : методические указания по выполнению лабораторных работ / сост.: Е. Б. Дунина, А. С. Соколова, А. А. Корниенко – Витебск : УО «ВГТУ», 2025. – 52 с.

В методических указания изложены теоретические сведения и задания к лабораторным работам по дисциплине «Разработка мультимедийных приложений». Издание предназначено для студентов специальностей 1-19 01 01-06 «Дизайн виртуальной среды» и 6-05-0211-05 «Графический дизайн и мультимедиадизайн (профилизация: Мультимедиадизайн)».

**УДК 004.9**
**ББК 32.97**

© УО «ВГТУ», 2025

# СОДЕРЖАНИЕ

# ЛАБОРАТОРНАЯ РАБОТА 1. C#: ТИПЫ ДАННЫХ

**Цель работы**: изучить классификацию типов данных и особенности синтаксических конструкций языка C#.

### *Краткие теоретические сведения*

Язык C# относится к строго типизированным языкам – любые используемые в нем идентификаторы должны принадлежать к какому-то типу. Для данных тип определяет способ их внутреннего (машинного) представления, а также совокупность действий, которые можно осуществлять над данными. Однако в C# типами считаются также классы и структуры, которые представляют собой совокупности данных и программного кода.

В отличие от общей системы типов .NET Framework (CTS), в C# имеются не только значащие и ссылочные типы, но также примитивные типы. В отличие от последних, примитивные типы обозначаются зарезервированными словами. Их можно использовать без ссылки на пространство имен.

Тип данных определяет:

- внутреннее представление данных;
- множество возможных значений;
- допустимые действия над данными.


Рисунок 1.1 – Различные классификации типов данных

К встроенным типам данных относятся: булевский, целый, символьный, Object, вещественный, строковый, финансовый.


Рисунок 1.2 – Основная классификация типов данных C#

Таблица 1.1 – Список числовых типов в C#

| Название | Ключевое слово | Класс-тип .Net | Диапазон значений | Размер, бит |
|---|---|---|---|---|
| Целочисленный со знаком | sbyte | sByte | от -128 до 128 | 8 |
| | short | Int16 | от -32768 до 32767 | 16 |
| | int | Int32 | от -2147483 до 2147483647 | 32 |
| | long | Int64 | от -92223372036854775808 до 92223372036854775807 | 64 |
| Целочисленный без знака | byte | Byte | от 0 до 255 | 8 |
| | ushort | UInt16 | от 0 до 65535 | 16 |
| | uint | UInt32 | от 0 до 4294967295 | 32 |
| | ulong | UInt64 | от 0 до 18446744073709551615 | 64 |
| С плавающей запятой | float | Single | от (+-)$1{,}5\times10^{-45}$ до $3{,}4\times10^{38}$ | 32 |
| | double | Double | от (+-)$5\times10^{-324}$ до $1{,}7\times10^{308}$ | 64 |
| Десятичное число (финансовый) | decimal | Decimal | от $-7{,}9\times10^{-28}$ до $7{,}9\times10^{28}$ | 128 |

Диапазон decimal меньше, чем у типа double, однако выше точность представления знаков в дробной части, что необходимо для реализации финансовых операций (точность 28-29 значащих цифр дробной части).

Бит – это самая маленькая единица измерения цифровой информации, которая может принимать значения 0 или 1. Байт – это 8 битов, объединённых в группу для удобства учёта.

Таблица 1.2 – Логический тип

| Название | Диапазон значений | Размер, бит |
|---|---|---|
| bool | true/false | 8 |

Таблица 1.3 – Символьный тип

| Название | Диапазон значений | Размер, бит |
|---|---|---|
| char | 0..65535 | 16 |

Таблица 1.4 – Строковый тип

| Название | Диапазон значений | Размер, бит |
|---|---|---|
| string | размер определяется числом символов строки | 16 на символ |

*Преобразование типа*. Автоматическое (неявное) преобразование возможно не всегда, а только если при этом не может произойти потеря значимости.

Явное преобразование типа можно задать с помощью:

1. Операции `(тип)x`. Например:

```
long b = 300;
int a = (int)b;
```

2. Методов класса `Convert`.

```
string s= “123”;
Convert.ToInt32(s);
```

3. Методов `Parse`.

```
Int.Parse(s);
```

На рисунке 1.3 показаны неявные арифметические преобразования типов. При отсутствии линии возникает ошибка компиляции.

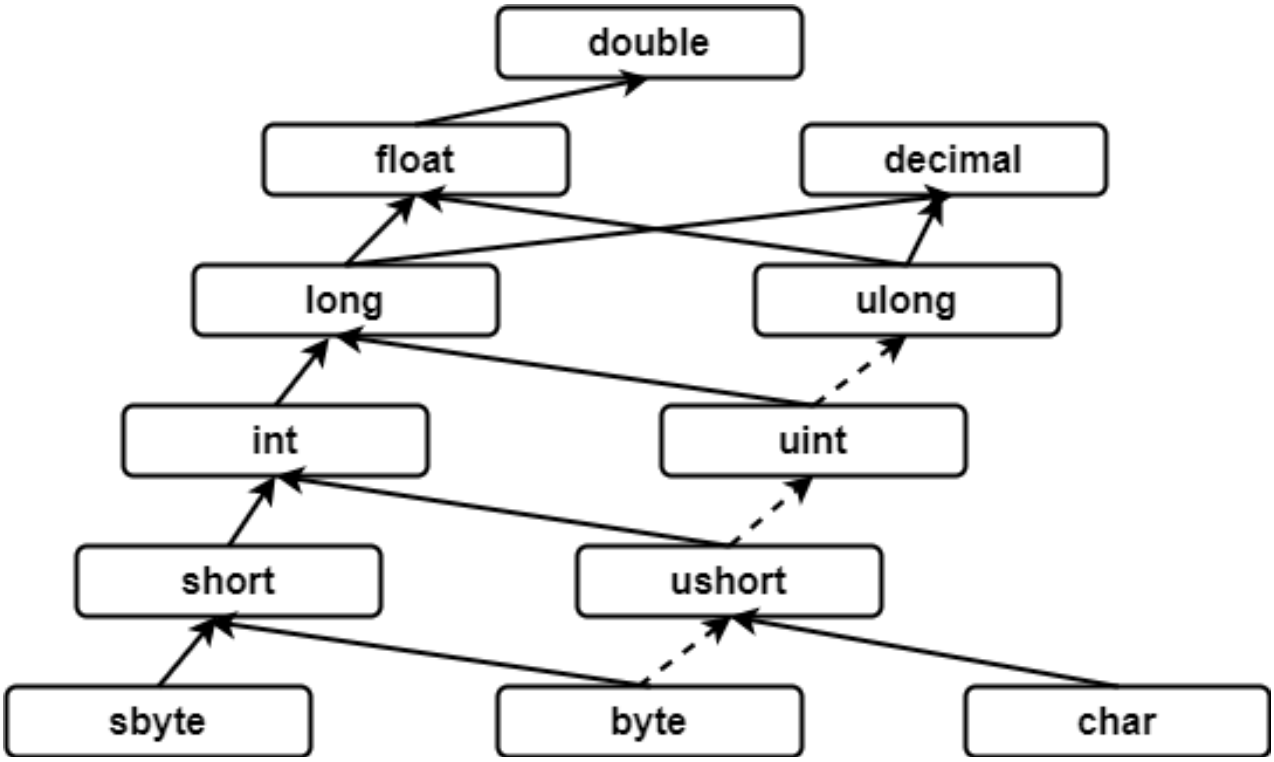


Рисунок 1.3 – Неявные арифметические преобразования типов

***Порядок выполнения работы***

1. Разработать приложение, позволяющее проверить принадлежность вводимого числа определенному числовому типу.

Пользовательский интерфейс представлен на рисунке 1.4.

Основные компоненты программы: три текстовых поля: `textBoxType`, `textBoxValue`, `textBox4`, `textBoxResult`; три кнопки: `buttonInput`, `buttonResult`, `btnClear`; четыре `label` и два `groupBox`.

2. Проанализировать код листинга 1.1 и создать соответствующие обработчики событий.

3. Добавить в обработчик события кнопки buttonInput код для анализа числовых типов: `uhort`, `uint`, `int`, `long`, `ulong`, `float`, `double`, `decimal`.

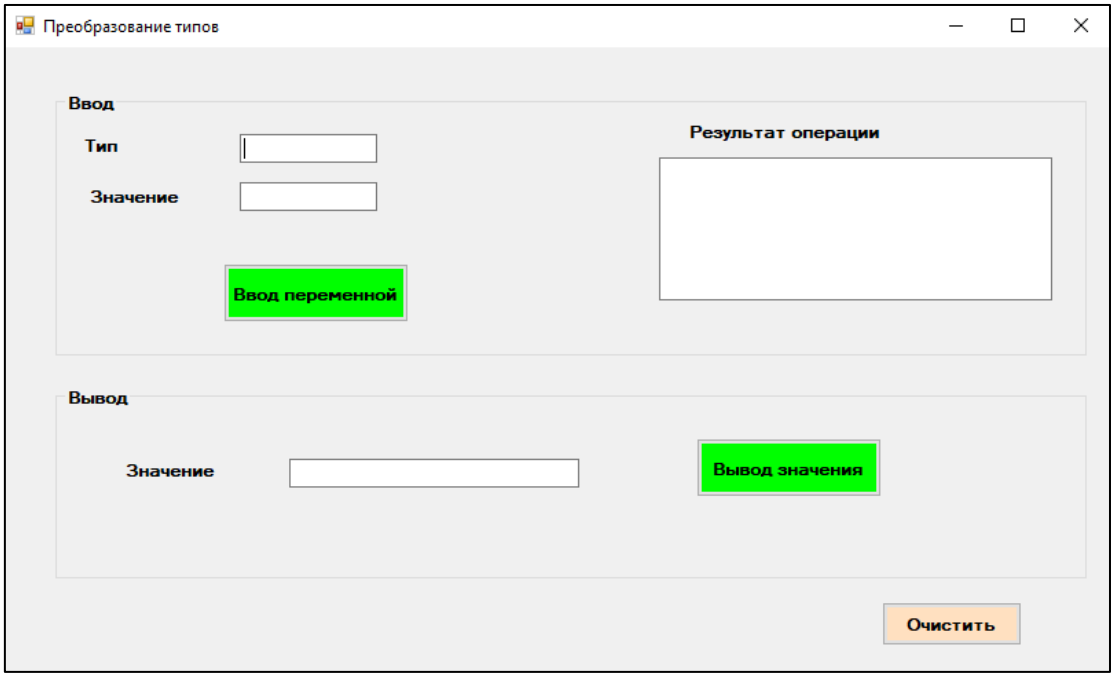


Рисунок 1.4 – Рабочее окно программы

Листинг 1.1 – Пример работы оператора выбора

```
public partial class Form1 : Form
{
    string strType = "";
    string strValue = "";
    string strResult = "";
    const string OK_MESSAGE = "Операция ввода прошла успешно";
    public const string ERR_MESSAGE = "Значение заданное при вводе, не
принадлежит типу ";
    const string ERR_Type_MESSAGE = "Неверно задан скалярный тип!";

    public Form1()
    {
        InitializeComponent();
        //textBoxType.Select();
    }
    private void buttonInput_Click(object sender, EventArgs e)
    {
        strType = textBoxType.Text;
        strValue = textBoxValue.Text;
        //выбор вариантов
```

```
        switch (strType)
        {
            case "byte":
            {
                byte x;
                try
                {
                    x = Convert.ToByte(strValue);
                    textBoxResult.Text = OK_MESSAGE;
                    strResult = x.ToString();

                    catch (Exception)
                    {
                        textBoxResult.Text = ERR_MESSAGE + "byte!";
                    }
                    break;
                }
            case "bool":
            {
                bool x;
                try
                {
                    x = Convert.ToBoolean(strValue);
                    textBoxResult.Text = OK_MESSAGE;
                    strResult = x.ToString();
                }
                catch (Exception)
                {
                    textBoxResult.Text = ERR_MESSAGE + "bool!";
                }
                break;
            }
            default:
            {
                textBoxResult.Text = ERR_Type_MESSAGE;
                break;
            }
        }
    }

    private void buttonResult_Click(object sender, EventArgs e)
    {
        if (textBoxResult.Text == ERR_MESSAGE + "byte!" || textBoxResult.Text
== ERR_MESSAGE + "bool!"
        || textBoxResult.Text == ERR_MESSAGE + "sbyte!" || textBoxResult.Text
== ERR_MESSAGE + "short!"
        || textBoxResult.Text == ERR_MESSAGE + "uint!" || textBoxResult.Text ==
ERR_MESSAGE + "int!"
        || textBoxResult.Text == ERR_MESSAGE + "long!" || textBoxResult.Text ==
ERR_MESSAGE + "ulong!"
        || textBoxResult.Text == ERR_MESSAGE + "float!" || textBoxResult.Text
== ERR_MESSAGE + "double!"
        || textBoxResult.Text == ERR_MESSAGE + "decimal!")
        { textBox4.Text = "ошибка ввода"; }
        else textBox4.Text = strResult;
    }
```

```
    private void btnClear_Click(object sender, EventArgs e)
    {
        textBoxType.Text = "";
        textBoxValue.Text = "";
        textBox4.Text = "";
        textBoxResult.Text = "";
    }
}
```

### *Содержание отчета*

Предоставить программу, написанную в Visual Studio на языке C# и результаты ее работы.

### *Контрольные вопросы*

1. На какие группы и разновидности разделяются все типы данных в C#? Привести примеры.
2. Основное отличие структурных типов от ссылочных?
3. Перечислить все целочисленные арифметические типы данных в C#. Указать их диапазон и размер занимаемой памяти.
4. Перечислить все вещественные арифметические типы данных в C#. Указать их диапазон и размер занимаемой памяти.

## ЛАБОРАТОРНАЯ РАБОТА 2. КЛАСС ENUM

**Цель работы**: изучить Enum в C# и некоторые его особенности.

### *Краткие теоретические сведения*

Тип перечисление (`enum`) представляет собой набор символьных имен, связанных с уникальными целочисленными значениями. Для объявления этого типа используется оператор enum. После оператора enum указывается имя создаваемого типа. Через двоеточие можно указать тип констант перечисления (должен быть целочисленным). Если не указывать тип явно, то будет использован `int`. После этого приводится список символьных констант через запятую.

Каждому элементу перечисления присваивается целочисленное значение. Эти значения начинаются с нуля и, далее, увеличиваются на единицу. Можно явно задать значения элементам перечисления. Для этого C# предоставляет два варианта:

– задание численного значения первому элементу, при этом все остальные, по очереди, принимают значения на единицу больше;

– явное задание численных значений каждому элементу перечисления индивидуально.

С элементами типа перечисления можно производить различные операции:

– приведение значения к целочисленному типу;
– операции сравнения (==, !=, <, > и т.п.);
– арифметические операции: сложение и вычитание;
– логические операции (&, |, !);
– операции постфиксного и префиксного инкремента (++) и декремента (--).

```
class Program
{
    Ссылок: 2
    enum WeekDay
    {
        Monday, Tuesday, Wednesday, Thursday, Friday, Saturday, Sunday
    }
    Ссылок: 0
    static void Main(string[] args)
    {
        WeekDay wd ;
        int dayNum;
        wd = WeekDay.Friday;
        Console.WriteLine($"Пример дня недели: {wd}");
        dayNum = (int)wd;
        Console.WriteLine($"{wd} идет в перечислении под номером {dayNum}");
        wd -= 3;
        dayNum = (int)wd; Console.WriteLine($"{wd} идет в перечислении под номером {dayNum}");
        Console.WriteLine();
        Console.ReadLine();
емы не найдены.
E:\1_task network\РМП_ДЗв\лекции my\решения к лекции\ENUM\ENUM\bin\Debug\ENUM.exe
Пример дня недели: Friday
Friday идет в перечислении под номером 4
Tuesday идет в перечислении под номером 1
```

Рисунок 2.1 – Работа с переменными типа перечисления

***Порядок выполнения работы***

1. Создадим новый проект. Пользовательский интерфейс представлен на рисунке 2.2.

Основные компоненты программы: два textbox, comboBox, кнопку – `btnCalculate`, четыре `label` (`lblResult` для вывода результата). Для `comboBox` в коллекцию `items` введите строки `Add`, `Substract`, `Multiply`, `Divide` (одну на строку).

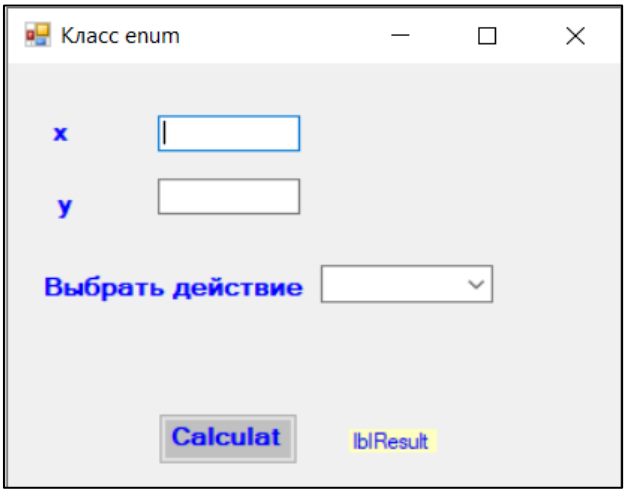


Рисунок 2.2 – Интерфейс формы

2. Часто переменная перечисления выступает в качестве хранилища состояния, в зависимости от которого производятся некоторые действия.

Определим перечисление `Operation`, которое представляет арифметические операции. Каждый тип операций определен в виде одной из констант перечисления.

Для этого создайте класс `Operation`. Замените слово `class` на `enum` (листинг 2.1).

Листинг 2.1 – Class Operation

```
public enum Operation
{
    Add,
    Substract,
    Multiply,
    Divide
}
```

3. В главной форме определен метод `ApplyOperation()`, который в качестве параметров принимает два числа и тип операции в виде константы перечисления и в зависимости от этого типа возвращает из конструкции `switch` результат определенной операции.

Проанализировать код и создать соответствующие обработчики событий (листинг 2.2).

Листинг 2.2 – Class Form1

```
public partial class Form1 : Form
{
    public Form1()
    {
        InitializeComponent();
    }

    public static void ApplyOperation(double x, double y, Operation op, out
double rez)
    {
        double result = 0.0;
        switch (op)
        {
            case Operation.Add: result = x + y; break;
            case Operation.Substract: result = x - y; break;
            case Operation.Multiply: result = x * y; break;
            case Operation.Divide: result = x / y; break;
        }
        rez = result;
    }

    private void btnCalculate_Click(object sender, EventArgs e)
    {
        double x = Convert.ToDouble(textBox1.Text);
        double y = Convert.ToDouble(textBox2.Text);
        string op1 = comboBox1.SelectedItem.ToString();
```

```
        if (op1 == "Add")
        {
            ApplyOperation(x, y, Operation.Add, out double rez);
            lblResult.Text = Convert.ToString(rez);
        }
        else
        {
            if (op1 == "Substract")
            {
                ApplyOperation(x, y, Operation.Substract, out double rez);
                lblResult.Text = Convert.ToString(rez);
            }
            else
            {
                if (op1 == "Multiply")
                {
                    ApplyOperation(x, y, Operation.Multiply, out double rez);
                    lblResult.Text = Convert.ToString(rez);
                }
                else
                {
                    if (op1 == "Divide")
                    {
                        ApplyOperation(x, y, Operation.Divide, out double rez);
                        lblResult.Text = Convert.ToString(rez);
                    }
                }
            }
         }
    }
}
```

4. Обработайте исключения, для ввода данных x и y.
5. Протестировать программу.

### *Содержание отчета*

Предоставить приложение, написанное в Visual Studio на языке C# и шаблонах Windows Forms. Продемонстрировать работу приложения.

### *Контрольные вопросы*

1. Что представляет собой перечисление enum?
2. Какое числовое значение имеет константа `Season.Summer` в следующем перечислении?

```
enum Season
{
    Winter = 10,
    Spring,
    Summer,
    Autumn
}
```

# ЛАБОРАТОРНАЯ РАБОТА 3. КАЛЬКУЛЯТОР ИНДЕКСА МАССЫ ТЕЛА

**Цель работы**: научиться добавлять формы, а также осуществлять взаимодействие между двумя формами.

### *Краткие теоретические сведения*

Чтобы добавить еще одну форму в проект, нажмем на имя проекта в окне Solution Explorer (Обозреватель решений) правой кнопкой мыши и выберем Add(Добавить) → Windows Form. Дадим новой форме какое-нибудь имя, например, Form2.cs (рис. 3.1).

Пусть первая форма по нажатию на кнопку будет вызывать вторую форму. Создаем обработчик события нажатия кнопки. Вторая форма называется `Form2`, поэтому сначала мы создаем объект данного класса, а потом для его отображения на экране вызываем метод `Show` (листинг 3.1).

Листинг 3.1 – Class Form1

```
public partial class Form1 : Form
{
    Form2 form2;

    public Form1()
    {
        InitializeComponent();
        form2 = new Form2(this);
        //в Form2 передаем Form1
    }

    private void button1_Click(object sender, EventArgs e)
    {
        form2.Show();
        Hide();//скрывает Form1
    }
 }
```

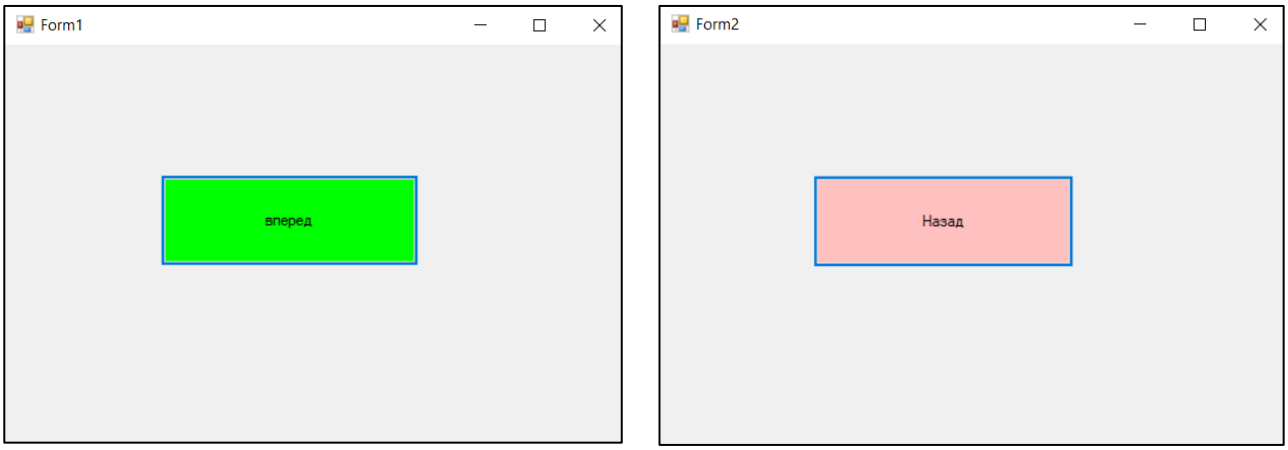


Рисунок 3.1 – Интерфейс Form1 и Form2

Для Form2 добавили новый конструктор public Form2(Form1 form1), в котором мы получаем первую форму.

Создаем обработчик события нажатия кнопки для Form2.

Листинг 3.2 – Class Form2

```
public partial class Form2 : Form
{
    Form1 form1;

    public Form2(Form1 form1)
    {
        InitializeComponent();
        //обращение к полю класса
        this.form1 = form1;
        //this  обозначает текущий экземпляр этого класса
    }

    private void button1_Click(object sender, EventArgs e)
    {
        form1.Show();
        Hide();
    }
}
```

При работе с несколькими формами надо учитывать, что одна из них является главной – та, которая запускается первой в файле Program.cs. Если у нас одновременно открыто несколько форм, то при закрытии главной закрывается приложение и вместе с ним все остальные формы.

### *Порядок выполнения работы*

1. Создайте калькулятор индекса массы тела.

На форме1 должны вводиться данные для расчета: Рост(см), Вес(кг), Пол (ж или м), кнопка «Вычислить».

На форме2 выводиться: индекс массы тела (ИМТ), пояснение к нему: «Согласно медицинским рекомендациям Ваш вес должен быть в пределах…, Ваш идеальный вес…».

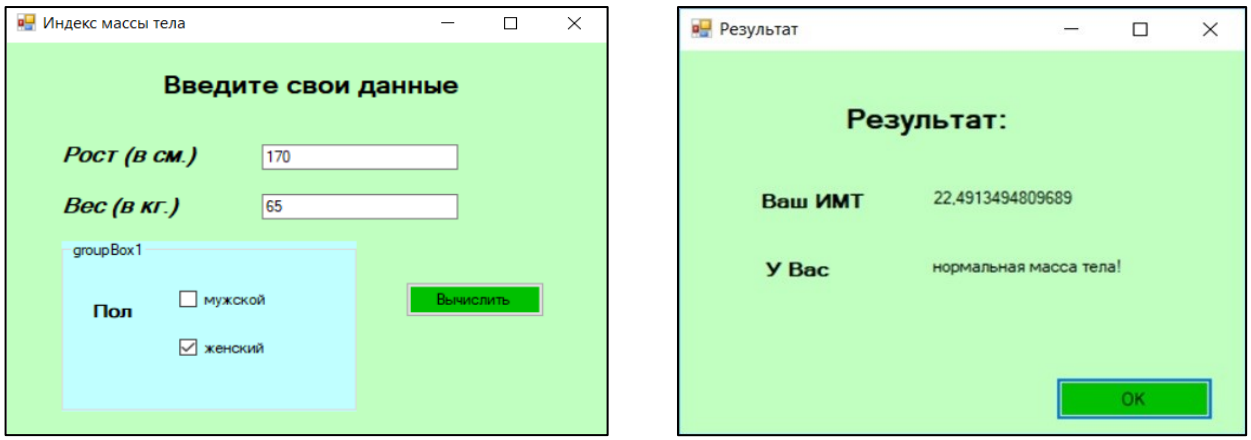


Рисунок 3.2 – Пример Form1 и Form2

При разработке приложения воспользуйтесь следующими сведениями.

Индекс массы тела ИМТ (англ. BMI) был предложен бельгийским ученым А. Кетле и может быть вычислен по следующей формуле:

$$ИМТ = \frac{m}{h^2},$$

где m – масса тела в килограммах, h – рост в метрах.

Для интерпретации значений рассчитанных индексов массы тела воспользуйтесь таблицей, приведенной ниже.

Таблица 3.1 – Индекс массы тела

| Индекс массы тела ИМТ | Классификация |
|---|---|
| Меньше 16 | Выраженный дефицит массы тела |
| 16–18,5 | Недостаточная масса тела |
| 18,5–25 | Нормальная масса тела |
| 25–30 | Избыточная масса тела |
| 30–35 | Ожирение 1-ой степени |
| 35–40 | Ожирение 2-ой степени |
| Больше 40 | Ожирение 3-ой степени |

2. Протестировать программу.

### *Содержание отчета*

Предоставить приложение «Индекс массы», написанное в Visual Studio на языке C# и шаблонах Windows Forms. Продемонстрировать его работу.

### *Контрольные вопросы*

1. Как открыть новую форму (`Form2`) из текущей формы (`Form1`) и закрыть текущую форму в Windows Forms?
2. Каковы различные способы передачи данных из `Form1` в `Form2` при переходе между формами в Windows Forms? Приведите пример кода для одного метода.
3. В чем разница между отображением формы в виде модального (`ShowDialog()`) и немодального (`Show()`) диалогового окна? Когда следует использовать каждый подход?

## ЛАБОРАТОРНАЯ РАБОТА 4. РАБОТА С НЕСКОЛЬКИМИ ФОРМАМИ

**Цель работы**: научиться работать с несколькими формами (Visual Studio).

### *Краткие теоретические сведения*

Рассмотрим пример работы приложения «Выберите снежинку, а мы расскажем вам удивительный факт о ней».

1. На Form1 поместите 6 кнопок. Правой мышью станьте на проект → Добавить → Создать папку (Images). Поместите в нее шесть рисунков. На каждую кнопку поместите один рисунок (свойства BackgroundImage и BackgroundImagLayout) (рис. 4.1).

2. Правой мышью станьте на проект → Добавить → Форма (Windows Forms) (Form2).

3. При нажатии на картинку (Form1) будет осуществляться переход на Form2. И в TextBox1 должно появляться одно из следующих сообщений с интересным фактов:

– 6 лучей имеют почти все снежинки. Их форма определяется строением молекулы воды $H_2O$: атом кислорода в центре, в вершинах – атомы водорода и неподеленные пары валентных электронов кислорода. При взаимодействии протонов одной молекулы с неподеленными электронами кислорода другой возникает водородная связь, удерживающая соседние молекулы под строго определенными углами (чаще всего 60°), которые дают шестиконечную форму. Снежинки с четырьмя, пятью и восемью лучами не встречаются, а вот с тремя и двенадцатью изредка находятся.

– 170 квинтиллионов (миллиард миллиардов). Из такого количества молекул воды состоит средняя снежинка. А вот ледяной куб в 10 $см^3$ содержит уже свыше 30 септиллионов (миллион миллиардов миллиардов) молекул ($3 \cdot 10^{25}$). Это превышает число звезд во Вселенной.

– В обычном стакане умещаются миллионы снежинок. Если все молекулы из этого стакана выложить в один ряд, то он многократно превысит расстояние от Солнца до Нептуна.

– 38 см – диаметр крупнейшей снежинки, которую, согласно Книге рекордов Гиннесса, нашли 28 января 1887 года в Форт-Кио (штат Монтана, США). Ее толщина – 20 см. Диаметр обычной снежинки не превышает половины сантиметра.

– Самые крупные снежинки в Москве выпали 30 апреля 1944 года. Они имели диаметр 10 см и напоминали по форме страусиные перья.

– 0,8 м/с – скорость, с которой падает стандартная снежинка в тихую погоду.

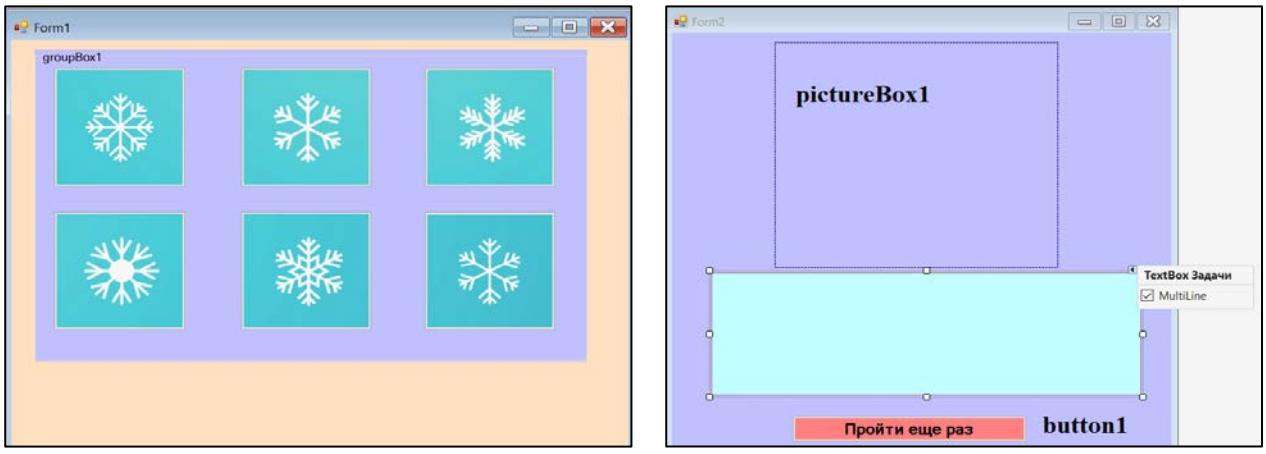


Рисунок 4.1 – Вид Form1 и Form2

Листинг 4.1 – Примерный код Form1.cs

```
public partial class Form1 : Form
{
    Form2 form2;
    public int t;

    public Form1()
    {
        InitializeComponent();
        form2 = new Form2(this);
        //в Form2 передаем Form1
    }

    private void button1_Click(object sender, EventArgs e)
    {
        form2.Show();
        t = 1;
        form2.UpdateNB();
        Hide();//скрывает Form1
    }

    private void button2_Click(object sender, EventArgs e)
    {
        form2.Show();
        t = 2;
        form2.UpdateNB();
        Hide();//скрывает Form1
    }

    private void button3_Click(object sender, EventArgs e)
    {
        form2.Show();
        t = 3;
        form2.UpdateNB();
        Hide();//скрывает Form1
    }

    private void button4_Click(object sender, EventArgs e)
    {
        form2.Show();
        t = 4;
        form2.UpdateNB();
        Hide();//скрывает Form1
    }

    private void button5_Click(object sender, EventArgs e)
    {
        form2.Show();
        t = 5;
        form2.UpdateNB();
        Hide();//скрывает Form1
    }

    private void button6_Click(object sender, EventArgs e)
    {
        form2.Show();
```

```
            t = 6;
            form2.UpdateNB();
            Hide();//скрывает Form1
        }
    }
```

Листинг 4.2 – Примерный код Form2.cs

```
public partial class Form2 : Form
{
    Form1 form1;
    int T;

    public Form2(Form1 form1)
    {
        InitializeComponent();
        this.form1 = form1;
        //this обозначает текущий экземпляр этого класса
    }

    public void UpdateNB()
    {
        T = form1.t;
        if (T == 1) {
            pictureBox1.Image = Image.FromFile("Images\\snowflake" + T + ".png");
            textBox1.Text = "6 лучей имеют почти все снежинки." +
                " Их форма определяется строением молекулы воды. Снежинки с" +
                " 4, 5, 8 лучами не встречаются, а с 3 и 12 изредка находятся";
        }
        else if (T == 2) {
            pictureBox1.Image = Image.FromFile("Images\\snowflake" + T + ".png");
            textBox1.Text = "170 квинтиллионов (миллиард миллиардов)" +
                " Из такого количества молекул воды состоит средняя снежинка.";
        }
        else if (T == 3) {
            pictureBox1.Image = Image.FromFile("Images\\snowflake" + T + ".png");
            textBox1.Text = "В обычном стакане умещаются миллионы снежинок. " +
                " Если все молекулы из этого стакана выложить в один ряд," +
                " то он многократно превысит расстояние от Солнца до Нептуна.";
        }
        else if (T == 4) {
            pictureBox1.Image = Image.FromFile("Images\\snowflake" + T + ".png");
            textBox1.Text = "38 см – диаметр крупнейшей снежинки, которую," +
                " согласно Книге рекордов Гиннесса, нашли 28 января 1887 " +
                "года в Форт-Кио (штат Монтана, США).";
        }
        else if (T == 5) {
            pictureBox1.Image = Image.FromFile("Images\\snowflake" + T + ".png");
            textBox1.Text = "Самые крупные снежинки в Москве выпали 30 апреля 1944" +
            " года. Они имели диаметр 10 см и напоминали по форме страусиные перья.";
        }
        else if (T == 6) {
            pictureBox1.Image = Image.FromFile("Images\\snowflake" + T + ".png");
            textBox1.Text = "0.8 м/с - скорость, с которой падает стандартная" +
                " снежинка в тихую погоду";
        }
    }
```

```
    private void Form2_FormClosed(object sender, FormClosedEventArgs e)
    {
        form1.Close();
        // закрытие главной формы
    }

    private void button1_Click(object sender, EventArgs e)
    {
        form1.Show();
        Hide();
    }
}
```

### *Порядок выполнения работы*

1. Реализовать пример, описанный в краткой теории.
2. Создать новый проект. Выберите свою тематику и разработайте аналогичное приложение.

### *Содержание отчета*

Предоставить два проекта, написанных в Visual Studio на языке C# и результаты их работы.

### *Контрольные вопросы*

1. Как можно предотвратить закрытие формы, когда пользователь нажимает кнопку «X», и вместо этого скрыть ее для последующего использования?
2. Как можно извлечь данные из Form2 обратно в Form1 после закрытия Form2, когда она была открыта с помощью `ShowDialog()`?
3. Каковы наилучшие практики управления несколькими открытыми формами в приложении Windows Forms, чтобы избежать утечек памяти?

## ЛАБОРАТОРНАЯ РАБОТА 5. ДВУМЕРНЫЕ МАССИВЫ

**Цель работы**: научиться использовать двумерные массивы в программах на языке C#.

### *Краткие теоретические сведения*

Массив – именованная упорядоченная последовательность однотипных элементов, к каждому из них можно обратиться по индексу.

Элементы массива имеют одно общее имя.

Виды массивов в C#:

- одномерные (линейные)
- многомерные (например, двумерные, или прямоугольные)
- массивы массивов (`jagged` – используются термины: ступенчатые, невыровненные, изрезанные, зубчатые…).

Описание массива:

- массив относится к ссылочным типам данных (располагается в хипе), поэтому создание массива начинают с выделения памяти под его элементы;
- элементами массива могут быть величины как значимых, так и ссылочных типов (в том числе массивы);
- элементы массивов значимых типов хранят значения, массивы ссылочных типов – ссылки на элементы;
- всем элементам при создании массива присваиваются значения по умолчанию: `0` – для значимых типов и `null` – для ссылочных.

Действия с массивами:

- с элементами массива можно делать все, что допустимо для переменных того же типа;
- при работе с массивом автоматически выполняется контроль выхода за его границы: если значение индекса выходит за границы массива, генерируется исключение `IndexOutOfRangeException`;
- массивы одного типа можно присваивать друг другу. При этом происходит присваивание ссылок, а не элементов:

```
int[] c = new int[10]; int[] b = c; // b и c указывают на один и тот же массив.
```

Все массивы в C# имеют общий базовый класс `Array`, определенный в пространстве имен `System`. Некоторые свойства и методы класса `Array`:

- `Length` (свойство) – количество элементов массива (по всем размерностям);
- `BinarySearch` (статический метод) –двоичный поиск в отсортированном массиве;
- `IndexOf` – (статический метод) – поиск первого вхождения элемента в одномерный массив;
- `Sort` (статический метод) – упорядочивание элементов одномерного массива;
- `Reverse` – изменение порядка следования элементов на обратный.

Варианты описания двумерного массива:

```
тип[,] имя;
тип[,] имя = new тип [ разм_1, разм_2 ];
тип[,] имя = { списки _значений };
тип[,] имя = new тип [,] { списки_значений };
тип[,] имя = new тип [ разм_1, разм_2 ] { списки_значений };
```

Примеры описаний:

```
int[,] a;     // элементов нет
int[,] b = new int[2, 3]; // элементы равны 0
int[,] c = { {1, 2, 3}, {4, 5, 6} }; // new подразумевается
int[,] c = new int[,] { {1, 2, 3}, {4, 5, 6} }; // размерность вычисляется
```

### *Порядок выполнения работы*

1. Создать новый проект «Windows-приложение вычисления суммы, разности матриц, умножения матрицы на число и транспонирование матрицы».

1.1. Пользовательский интерфейс представлен на рисунке 5.1 – 5.2.

Основные компоненты `Form1`: `comboBox`, 2 `dataGridView` (`dataGridViewA`, `dataGridViewB`), `button1-4` – для вычисления суммы,

разности, произведения матриц и умножения матрицы на число, а также 2 кнопки `bttnReadFromFile`, `bttnClear`, текстовое поле `textBoxNumber`, четыре `label`.

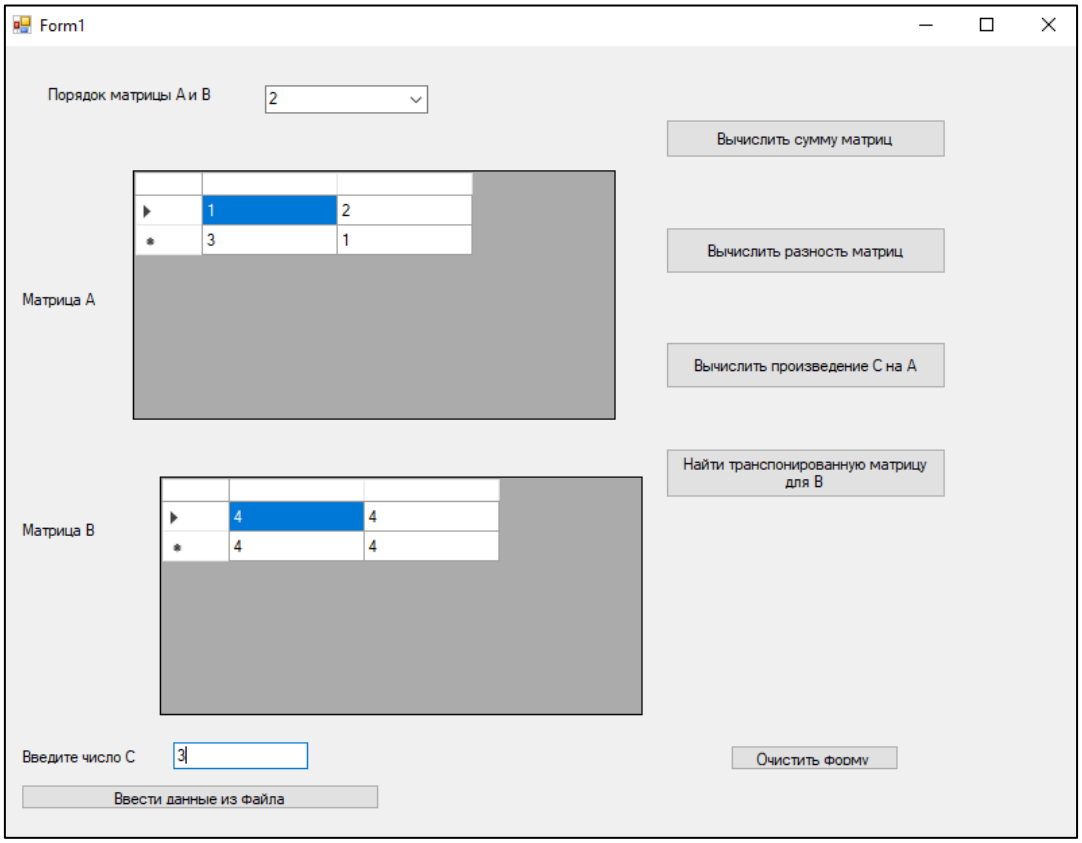


Рисунок 5.1 – Рабочее окно Form1

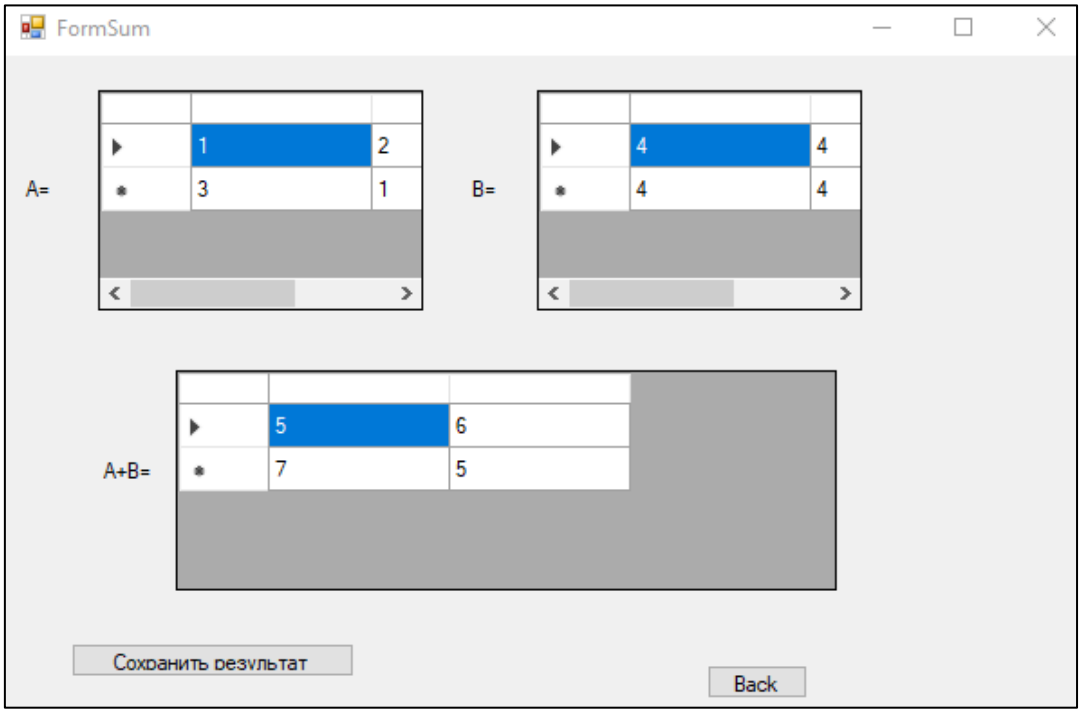


Рисунок 5.2 – Рабочее окно FormSum

Основные компоненты FormSum: три `dataGridView` (`dataGridViewA`, `dataGridViewB`, `dataGridViewSum`), 2 кнопки `bttnSaveResult`, `bttnBack`, три `label`.

1.2. Проанализировать код `Form1` и создать соответствующие обработчики событий (листинг 5.1).

Листинг 5.1 – Код Form1

```
public partial class Form1 : Form
{
    public int razmer;
    public double[,] matrixA;
    public double[,] matrixB;
    public double[,] matrixSum;
    FormSum formSum;
    FormDif formDif;
    FormMulN formMulN;
    FormTransp formTransp;

    public Form1()
    {
        InitializeComponent();
        formSum = new FormSum(this);
        formDif = new FormDif(this);
        formMulN = new FormMulN(this);
        formTransp = new FormTransp(this);
    }

    private void button1_Click(object sender, EventArgs e)
    {
        for (int i = 0; i < razmer; i++) {
            for (int j = 0; j < razmer; j++) {
                object cellValueA = dataGridViewA.Rows[i].Cells[j].Value;
                if (cellValueA != null && double.TryParse(cellValueA.ToString(),
out double a)) {
                    matrixA[i, j] = a;
                }
                else {
                    MessageBox.Show("Enter numbers correctly in matrix A.",
"Error.", MessageBoxButtons.OK, MessageBoxIcon.Information);
                    return;
                }
                object cellValueB = dataGridViewB.Rows[i].Cells[j].Value;
                if (cellValueB != null && double.TryParse(cellValueB.ToString(),
out double b)) {
                    matrixB[i, j] = b;
                }
                else {
                    MessageBox.Show("Enter numbers correctly in matrix B.",
"Error.", MessageBoxButtons.OK, MessageBoxIcon.Information);
                    return;
                }
            }
        }
        for (int i = 0; i < razmer; i++) {
```

```
            for (int j = 0; j < razmer; j++)
            {
                matrixSum[i, j] = matrixA[i, j] + matrixB[i, j];
            }
        }
        formSum.UpdateSum();
        formSum.Show();
        Hide();
    }

    private void comboBox1_SelectedIndexChanged (object sender, EventArgs e)
    {
        razmer = Convert.ToInt32((comboBox1.SelectedItem.ToString()));
        dataGridViewA.RowCount = razmer;
        dataGridViewA.ColumnCount = razmer;
        dataGridViewB.RowCount = razmer;
        dataGridViewB.ColumnCount = razmer;
        matrixA = new double[razmer, razmer];
        matrixB = new double[razmer, razmer];
        matrixSum = new double[razmer, razmer];
    }

    private void Form1_Load(object sender, EventArgs e)
    {
        comboBox1.SelectedIndex = 0;
        dataGridViewA.RowCount = 2;
        dataGridViewA.ColumnCount = 2;
        dataGridViewB.RowCount = 2;
        dataGridViewB.ColumnCount = 2;
    }

    private void bttnClear_Click(object sender, EventArgs e)
    {
        for (int i = 0; i < razmer; i++)
            for (int j = 0; j < razmer; j++) {
                dataGridViewA.Rows[i].Cells[j].Value = string.Empty;
                dataGridViewB.Rows[i].Cells[j].Value = string.Empty;
                textBoxNumber.Text = "";
            }
    }

    private void bttnReadFromFile_Click(object sender, EventArgs e)
    {
        string strLineA, strLineB;
        StreamReader sr = new StreamReader("InDataMatrix.txt");
        strLineA = sr.ReadLine();
        string[] args = strLineA.Split(); //Метод String.Split создает массив
подстрок, разбивая входную строку по одному или нескольким разделителям.
        int k = 0;
        for (int i = 0; i < razmer; i++) {
            for (int j = 0; j < razmer; j++) {
                matrixA[i, j] = Convert.ToDouble(args[k]); k++;
                dataGridViewA.Rows[i].Cells[j].Value = Convert.ToDouble(matrixA[i, j]);
            }
        }
        strLineB = sr.ReadLine();
```

```
            string[] argsB = strLineB.Split();
            k = 0;
            for (int i = 0; i < razmer; i++) {
                for (int j = 0; j < razmer; j++) {
                    matrixB[i, j] = Convert.ToDouble(argsB[k]); k++;
                    dataGridViewB.Rows[i].Cells[j].Value = Convert.ToDouble(matrixB[i,j]);
                }
            }
        }
    }
```

1.2 Проанализировать код FormSum и создать соответствующие обработчики событий (листинг 5.2).

Листинг 5.2 – Код FormSum

```
public partial class FormSum : Form
{
    int raz;
    Form1 form1;

    public FormSum (Form1 form1)
    {
        InitializeComponent();
        this.form1 = form1;
    }

    public void UpdateSum()
    {
        raz = form1.razmer;
        dataGridViewA.RowCount = raz;
        dataGridViewA.ColumnCount = raz;
        dataGridViewB.RowCount = raz;
        dataGridViewB.ColumnCount = raz;
        dataGridViewSum.RowCount = raz;
        dataGridViewSum.ColumnCount = raz;
        for (int i = 0; i < raz; i++)
            for (int j = 0; j < raz; j++) {
                dataGridViewA.Rows[i].Cells[j].Value = form1.matrixA[i, j];
                dataGridViewB.Rows[i].Cells[j].Value = form1.matrixB[i, j];
                dataGridViewSum.Rows[i].Cells[j].Value = form1.matrixSum[i, j];
            }
    }

    private void FormSum_FormClosed(object sender, FormClosedEventArgs e)
    {
        form1.Close();
        // закрытие главной формы
    }

    private void bttnSaveResult_Click(object sender, EventArgs e)
    {
        StreamWriter outFile = new StreamWriter("OutN.txt", false);
        for (int i = 0; i < raz; i++) {
            for (int j = 0; j < raz; j++)  {
                outFile.Write(Convert.ToString(form1.matrixSum[i, j]) + " ");
```

```
            }
        }
        outFile.Close();
    }

    private void bttnBack_Click(object sender, EventArgs e)
    {
        form1.Show();
        Hide();
    }
}
```

1.3 Создать текстовый файл для считывания данных (рис. 5.3). Данный файл необходимо поместить в папку bin → Debug.


Рисунок 5.3 – Файл для считывания данных

1.4 Дополнить Windows-приложение вычислением разности матриц A и B, умножением матрицы на число и транспонированием матрицы. Данные считываются из файла и результат выводится в файл.

Для выполнения задания необходимо создать еще несколько форм (рис. 5.4).

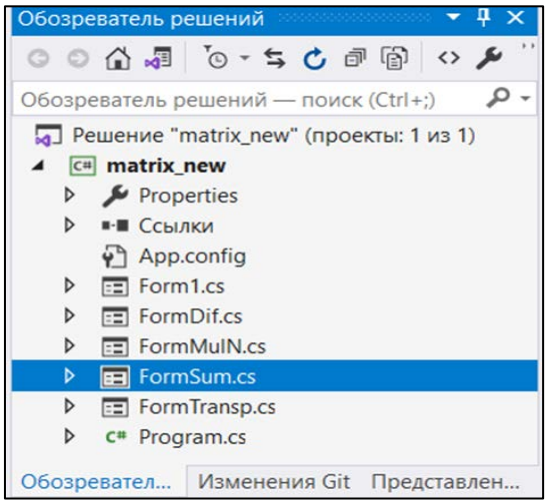


Рисунок 5.4 – Создание нескольких форм

2. Написать приложение (Windows Form) «Произведение двух матриц». Ввод матрицы в `dataGridView` пользователем и считывание из файла.

3. Написать приложение (Windows Form) «Вычисление определителя матрицы 2x2 и 3x3». Ввод матрицы в `dataGridView` пользователем и считывание из файла.

4. Написать приложение вычисляющие след матрицы. Ввод матрицы в `dataGridView` пользователем и считывание из файла.

След матрицы – сумма диагональных элементов матрицы. Это понятие устанавливается только для квадратной матрицы и обозначается через SpA (от нем. Spur – след) или TrA (от англ. trace – след). Таким образом:

$$TrA = a_{11} + a_{22} + ... + a_{nn}.$$

### *Содержание отчета*

Продемонстрировать работу четырех приложений: «Windows-приложение вычисления суммы, разности матриц, умножения матрицы на число и транспонирование матрицы», «Произведение двух матриц», «Вычисление определителя матрицы 2x2 и 3x3», «След матрицы».

### *Контрольные вопросы*

1. Как следует объявлять двумерный массив?
2. Как объявить и инициализировать двумерный массив целых чисел размером 3x3 в C#?
3. Напишите код, который заполняет двумерный массив N x M случайными числами от 1 до 100.
4. Как вывести двумерный массив в консоль в виде таблицы (со строками и столбцами)?
5. Напишите метод, который проверяет, содержится ли заданное число в двумерном массиве.
6. Как найти сумму всех элементов двумерного массива? А сумму элементов каждой строки?
7. Напишите код, который находит максимальный и минимальный элементы в двумерном массиве и их индексы.
8. Как транспонировать двумерный массив (поменять строки и столбцы местами)?
9. Как проверить, является ли двумерный массив (квадратная матрица) симметричным относительно главной диагонали?

## ЛАБОРАТОРНАЯ РАБОТА 6. ИГРА «ПЯТНАШКИ»

**Цель работы**: создание простейшей игры «Пятнашки» на языке C#.

### *Краткие теоретические сведения*

Пятнашки – одна из популярнейших в мире головоломок. Она представляет собой комплект, в который входит квадратная коробка, грань которой равняется 4 граням костяшки, то есть 4x4. В центре этой коробки 15 квадратных фишек. В коробке остается одно пустое место под одну фишку.

Задача игры – упорядочивание фишек по порядку. Концом игры будут считаться, собранные костяшки от 1 до 15 друг за другом.

В приложении чтобы сдвинуть «костяшку» из «пятнашек» в свободную ячейку, следует «кликнуть» по ней мышкой. Передвинуть подобным образом можно только «костяшку», которая расположена слева/справа или сверху/снизу относительно свободной ячейки.

*Решаемые и нерешаемые комбинации.* Начальную позицию можно разложить многими способами. Всего существует 16!=20 922 789 888 000 начальных позиций.

Это с учетом расположений пустой клетки. Если же рассмотреть, что пустая клетка всегда находится на одном месте (например, место 1 – левый верхний угол – или место 16 – правый нижний угол, пустая позиция), то получим 15!=1 307 674 368 000 позиций.

Половина из всех раскладов решается. Другая половина не собирается, так как приходит вот к такому положению:


Рисунок 6.1 – Задача Лойда

Пусть плитка с числом $i$ расположена до (если считать слева направо и сверху вниз) плиток с числами, меньшими $i$; тогда обозначим $n_i = k$. В частности, если после плитки с числом $i$ нет плиток с числами, меньшими $i$, то $n_i = 0$.

Также введём число $e$ – номер ряда пустой клетки (счет с 1). Если сумма

$$N = \sum_{i=1}^{15} n_i + e$$

нечётна, то решения головоломки не существует.

Рассмотрим пример. Возьмем следующий расклад:

Сначала берем первое число (у нас 12). Считаем, сколько чисел меньше находится после него. На примере это одиннадцать чисел (от 1 до 11 – эти числа меньше и расположены позже, то есть после рассматриваемого нами числа 12). Результат записали.

Теперь берем второе число (5). Считаем, сколько меньших чисел стоит после него. После пятерки только четыре числа младше (от 1 до 4). Результат прибавили к прошлому значению (11 + 4 = 15).

Теперь берем третье число (8) и считаем также количество меньших чисел, стоящих после (их шесть, то есть от 1 до 7, не считая 5, так как пятерка стоит раньше). Так и продолжаем дальше.

Не забываем в самом конце прибавить ряд с пустой клеткой. У нас на примере это третий ряд.

### *Порядок выполнения работы*

1. Создать новый проект. Пользовательский интерфейс представлен на рисунке 6.2.

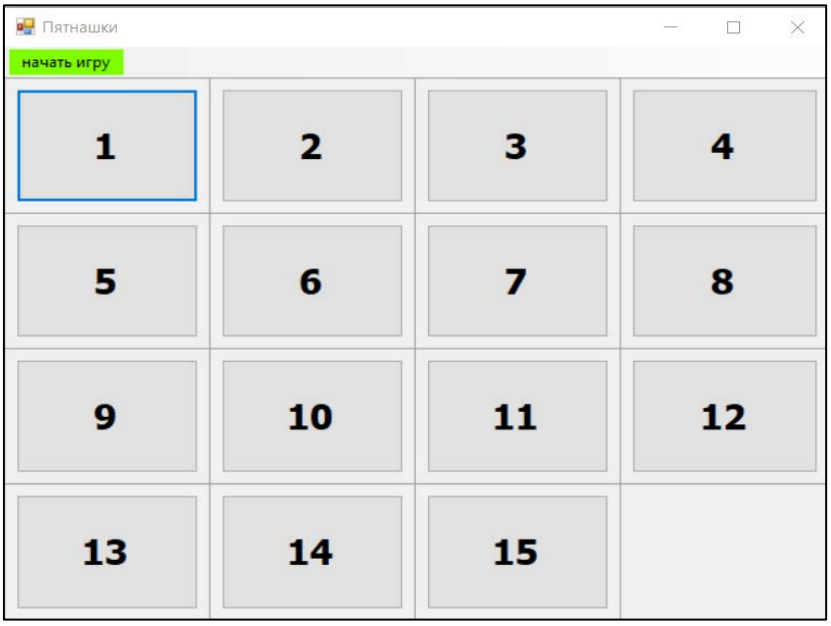


Рисунок 6.2 – Рабочее окно программы

2. Подготовка формы:

- сделаем, что бы форма запускалась в центре экрана (свойства → StartPosition → CenterScreen);

- добавим `MenuStript` на форму. Переименуем Name: `menu_start`. И наберем: `Начать игру`. При двойном нажатии будет создано событие `private void menu_start_Click(object sender, EventArgs e)` (листинг 6.1);

- переименуем форму (вместо `Form1` напишем `FormGame15`, → разместим на форме компонент `tableLayoutPanel`. Добавим `AddColumn`, `AddRow` до четырех столбцов и строк. Затем нажимаем `Edit Row and Columns`, выделяем все столбцы и указываем `Persent 50,00`;

- выделяем все строчки и поступаем аналогично;

- для tableLayoutPanel установить свойство Dock-Fill (на всю форму);

- для tableLayoutPanel установить свойство CellBorderStyle – Single → установим одну кнопку button1 → свойство Dock-Fill. Затем свойство Margin (расстояние от краев) – 10;10;10;10 (All 10). Text напишем прочерк «-». Name – button 0. Font – Tahoma, полужирный, размер 20;

- нажимаем Ctrl C и вставляем по порядку остальные кнопки (рис. 6.3).

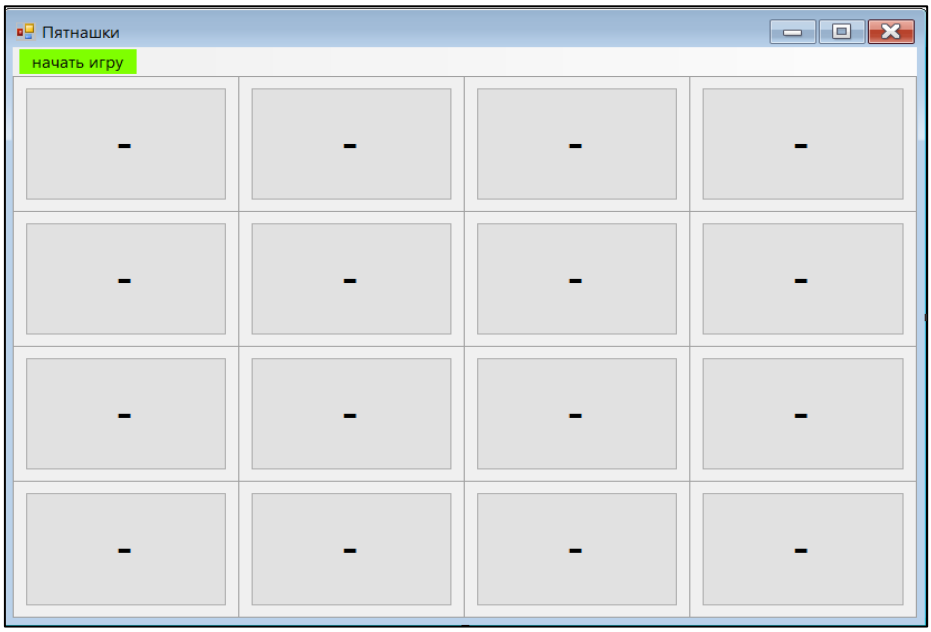


Рисунок 6.3 – Вид FormGame15

- для проверки запустите программу. Выделяем все кнопки и добавляем событие Click. Появится функция:

```
private void button15_Click(object sender, EventArgs e) {}
```

Она вызывается при нажатии на любую кнопку;

- пропишем номера кнопок. Для этого для каждой кнопки в свойстве Tag пропишем 0, для следующей 1 и т. д. до 15. Информация какая кнопка нажата содержится в object sender.

- перемешивание плашек будет ни случайным образом, а мы будем делать много ходов. Это позволит избежать неразрешимости задачи. Код приведен в методе private void start_game() (листинг 6.1);

Листинг 6.1 – Класс FormGame15

```
public partial class FormGame15 : Form
{
    Game game;

    public FormGame15()
    {
        InitializeComponent();
        game = new Game(4);
    }
```

```
private void button15_Click(object sender, EventArgs e)
{
    //узнаем какая кнопка нажата
    int position = Convert.ToInt16(((Button)sender).Tag);
    game.shift(position);
    refresh();
    if (game.check_numbers()) {
        MessageBox.Show("Вы победили!");
        //начинаем новую игру
        start_game();
    }
}

private Button button(int position)
{
    //метод возвращает кнопку, которая нам нужна
    switch (position)
    {
        case 0: return button0;
        case 1: return button1;
        case 2: return button2;
        case 3: return button3;
        case 4: return button4;
        case 5: return button5;
        case 6: return button6;
        case 7: return button7;
        case 8: return button8;
        case 9: return button9;
        case 10: return button10;
        case 11: return button11;
        case 12: return button12;
        case 13: return button13;
        case 14: return button14;
        case 15: return button15;
        default: return null;//если никакая кнопка не нажата
    }
}

private void menu_start_Click(object sender, EventArgs e)
{
    start_game();
}

private void start_game()
{
    game.start();
    //делаем перемешивание, например, ставим 100
    for (int j = 0; j < 10; j++)
        game.shift_random();
    refresh();
}

private void refresh()
//обнавляет содержимое кнопок
{
    for (int position = 0; position < 16; position++) {
```

```
            int nr = game.get_number(position);
            button(position).Text = nr.ToString();
            button(position).Visible = (nr > 0);
        }
    }

    private void FormGame15_Load(object sender, EventArgs e)
    {
        start_game();
    }
}
```

3. Создать класс Game.cs (для этого становимся на проект и нажимаем правую клавишу мыши → добавить → класс). В нем храниться вся логика игры.

Следует отметить, что кнопки имеют следующие координаты

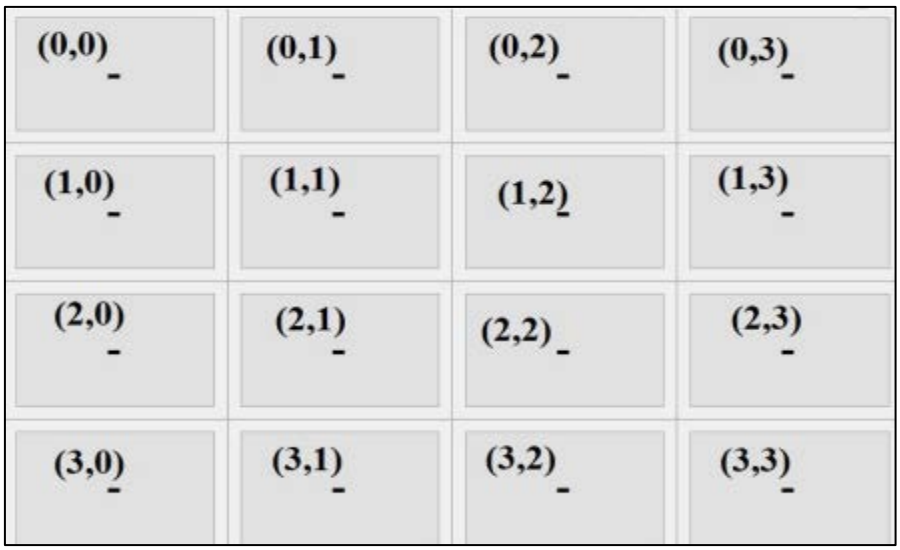


Рисунок 6.4 – Координаты кнопок

- кнопки пронумерованы по позиции 0, 1, 2, 3, …, 15. Поэтому создадим функции для перевода позиций в координаты и перевода координат в позицию. Это методы `public int get_number (int position)` и `private int coords_to_position (int x, int y)` (листинг 6.2).

4. Реализация перемещения кнопок. Сами кнопки не двигаются, у них меняется только надпись. Это реализовано в методе `public void shift (int position)` и `public void shift_random()`.

Листинг 6.2 – Класс Game

```
class Game
//логика игры
{
    int size;
    int[,] map;
    int spase_x, spase_y;
    static Random rand = new Random();
```

```
public Game(int size)
{
    //передаем размер поля для универсальности игры
    if (size < 2) size = 2;
    if (size > 5) size = 5;
    this.size = size;//устанавливаем размер, который указан
    map = new int[size, size];
}

public void start()
{
    //функция подготовит поле к игре
    for (int x = 0; x < size; x++)
        for (int y = 0; y < size; y++)
            map[x, y] = coords_to_position(x, y) + 1;
        //+1 т.л. позиция начинается с 0 , а цифры начинаются с 1
        spase_x = size - 1; spase_y = size - 1;
        // в последней клетке должно быть пусто
        map[spase_x, spase_y] = 0; //это пробел (свободное место)
}

public void shift(int position)
{
    int x, y;
    position_to_coords(position, out x, out y);
    //проверяем расположение плашки рядом или нет
    if (Math.Abs(spase_x - x) + Math.Abs(spase_y - y) != 1)
        return;
    map[spase_x, spase_y] = map[x, y];
    map[x, y] = 0;
    spase_x = x;
    spase_y = y;
}

public void shift_random()
{
    int a = rand.Next(0, 4);
    int x = spase_x;
    int y = spase_y;
    switch (a)
    {
        case 0: x--; break;
        case 1: x++; break;
        case 2: y--; break;
        case 3: y++; break;
    }
    shift(coords_to_position(x, y));
}

public bool check_numbers()
{
    //функция закончена ли игра
    if (!(spase_x == size - 1 && spase_y == size - 1))
        return false;
    for (int x = 0; x < size; x++)
```

```
            for (int y = 0; y < size; y++)
                if (!(x == size - 1 && y == size - 1))
                    if (map[x, y] != coords_to_position(x, y) + 1)
                        return false;
        return true;
    }

    public int get_number(int position)
    {
        //возвращает число в нужном месте (что написано на кнопке)
        int x, y;
        position_to_coords(position, out x, out y);
        if (x < 0 || x > size) return 0;
        if (y < 0 || y > size) return 0;
        return map[x, y]; //возвращаем надпись
    }

    private int coords_to_position(int x, int y)
    {
        //переведем координаты в позицию
        //позиция - одно число (кнопки 0 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15)
        //координаты - два числа (координаты кнопок)
        //0,0 0,1 0,2 0,3
        //1,0 1,1 1,2 1,3
        //2,0 2,1 2,2 2,3
        //3,0 3,1 3,2 3,3
        if (x < 0) x = 0;
        if (x > size - 1) x = size - 1;
        if (y < 0) y = 0;
        if (y > size - 1) y = size - 1;
        return y * size + x;
    }

    private void position_to_coords(int position, out int x, out int y)
    {
        if (position < 0) position = 0;
        if (position > size * size - 1) position = size * size - 1;
        x = position % size;
        y = position / size;
    }
}
```

### *Содержание отчета*

Предоставить программу, написанную в Visual Studio на языке C# и результаты ее работы.

### *Контрольные вопросы*

1. Как представить игровое поле в коде?
2. Как реализовать перемещение фишек?
3. Как перемешать фишки в начале игры?
4. Как добавить MenuStrip в Windows Forms приложение?
5. Как обработать клик по пункту меню?
6. Как добавить иконки в MenuStrip?

# ЛАБОРАТОРНАЯ РАБОТА 7. СЕТЬ ХОПФИЛДА

**Цель работы**: создать приложения для распознавания и восстановления графических образов с помощью сети Хопфилда.

***Краткие теоретические сведения***

Сеть Хопфилда – это рекуррентная искусственная нейронная сеть, используемая для ассоциативной памяти и оптимизации. Она может хранить определённые паттерны (образы) и восстанавливать их по зашумлённым или неполным входным данным.

Сеть Хопфилда является однослойной сетью, так как в ней используется лишь один слой нейронов, каждый из которых связан со всеми остальными. Архитектура сети изображена на рисунке 7.1.

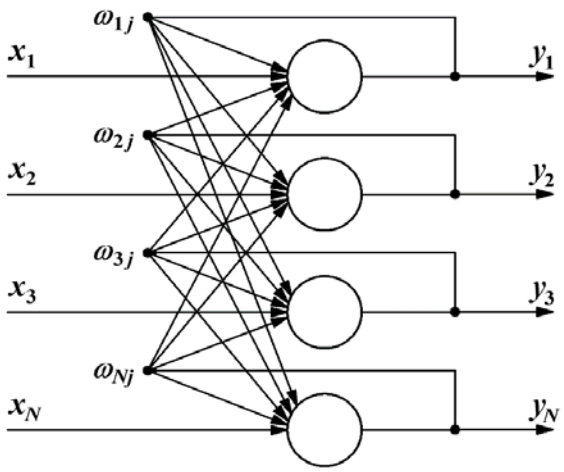


Рисунок 7.1 – Архитектура нейросети Хопфилда

Сеть Хопфилда – это искусственная нейронная сеть с шаговой функцией активации. Наиболее часто используется сигнум-функция:

$$sign(u) = f(u) = \begin{cases} 1, & \text{если } u \geq 0 \\ -1, & \text{если } u < 0 \end{cases}.$$

Особенностью сети Хопфилда является то, что она может распознать только те образы, которые ей представлены в процессе обучения в качестве тестовых данных. В сети Хопфилда после представления объектов обучения, матрица весов формируется сразу.

В начале обучения всем весам присваивают нулевые значения. Для всех i и j, которые меняются от 1 до N, выполняются следующие действия. Для всех образов выполняем корректировку весов:

$$W_{ij} = W_{ij} + x_i x_j .$$

Далее диагональные элементы зануляют:

$$W_{ii} = 0$$

и выполняют нормировку:

$$W_{ij} = \frac{W_{ij}}{N}.$$

Наша сеть обучена.

Далее матрица весов W не изменяется. Следует отметить, что для сети Хопфилда матрица весовых коэффициентов – симметрична $W_{ij} = W_{ji}$ и имеет нулевую главную диагональ $W_{ii} = 0$, т. е. отсутствует обратная связь нейрона на себя.

Процедуру распознавания образов можно продемонстрировать следующим алгоритмом. Допустим мы ищем образ:

$$y = (y_1; y_2; ...; y_N).$$

1. Цикл по $j$ от 1 до $N$:
2. Положим $d = 0$.
3. Вложенный цикл по $i$ от 1 до $N$:
4. $d = d + W_{ij} \cdot y_i$.
5. Конец вложенного цикла.
6. Если $d > 0$, то положить $z_j = 1$, иначе $z_j = -1$.
7. Конец внешнего цикла.
8. Получен вектор $Z = [Z_1, Z_2, ..., Z_N]$. Если вектор $Z$ содержится во множестве исходных образов, то считаем, что алгоритм нашел образ $Z$, который соответствует исходному образу $Y$.
9. Положим $Y = Z$.
10. Переход к шагу 1.

Пусть имеется нейронная сеть размерностью $N = n \times n$ нейронов, которые будут рассматриваться как квадратная матрица из n строк и n столбцов. Одним из недостатков сети Хопфилда – это относительно небольшой объем памяти, величину которой можно оценить выражением:

$$M \approx \frac{N}{2 \ln N}.$$

При записи большего числа образов нейронная сеть перестанет их распознавать.

### *Порядок выполнения работы*

1. Разработка программного продукта осуществляется на языке программирования C#. Приложение должно обеспечивать выполнение перечисленных ниже функций:

– обеспечение ввода «правильных» образов, с помощью которых должно происходить обучение системы;

– обеспечение возможности просмотра введённых «правильных» образов;

– построение матрицы весовых коэффициентов для сети Хопфилда;

– обеспечение ввода «испорченного» образа;

– нахождение «правильного» образа.

2. На форме размещаем необходимые элементы: 6 кнопок `Button` (`Name`: `btnAdd`, `btnShow`, `btnRestore`, `btnClearTable`, `btnClearImages`, `Exit`) и `pictureBox`.

3. Пример формы и кода можно посмотреть в лабораторной работе 5 пособия: Дунина, Е. Б. Искусственный интеллект. Лабораторный практикум: учебно-методическое пособие для специальности 1-40 05 01 «Информационные системы и технологии (по направлениям)» направления специальности 1-40 05 01-01 «Информационные системы и технологии (в проектировании и производстве)» / Е. Б. Дунина, А. С. Соколова, А. А. Корниенко; УО «ВГТУ». – Витебск, 2024. – 88 с.

### *Содержание отчета*

Предоставить программу, написанную в Visual Studio на языке C# и результаты ее работы.

### *Контрольные вопросы*

1. Как работает сеть Хопфилда?
2. Какие основные применения сети Хопфилда?
3. Как обучается сеть Хопфилда?
4. Какие ограничения у сети Хопфилда?
5. Как сеть Хопфилда восстанавливает образы?

## ЛАБОРАТОРНАЯ РАБОТА 8. WINDOWS FORMS И БАЗА ДАННЫХ MY SQL

**Цель работы**: изучить возможности взаимодействия с базой данных MySql.

### *Краткие теоретические сведения*

Создание базы данных MySql и подключение к ней.

Необходимо зайти в Program Files(x86). Находим папку MySql → Connector NET 8.0 → Assemblies → v4.5.2-MySql.Data.dll. Переходим в Visual

Studio, находим Ссылки. Кликаем правой мышью → Добавить ссылку (рис. 8.1). Далее переходим в Обзор и находим файл MySql.Data.dll (рис. 8.2), нажимаем Ok. В ссылках появиться MySql.Data (рис. 8.3).

Благодаря этой ссылки можно использовать различные классы, для подключения к базе данных.

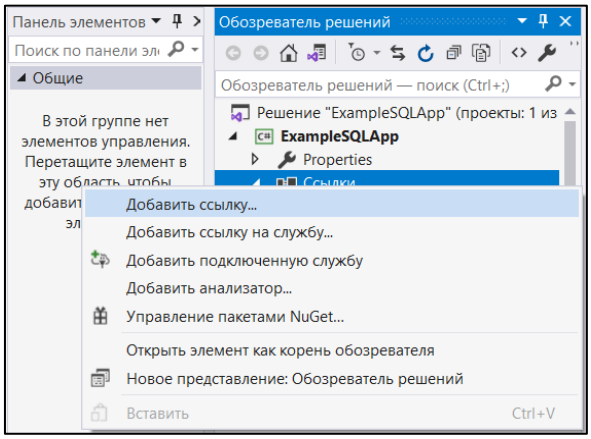


Рисунок 8.1 – Добавление ссылки

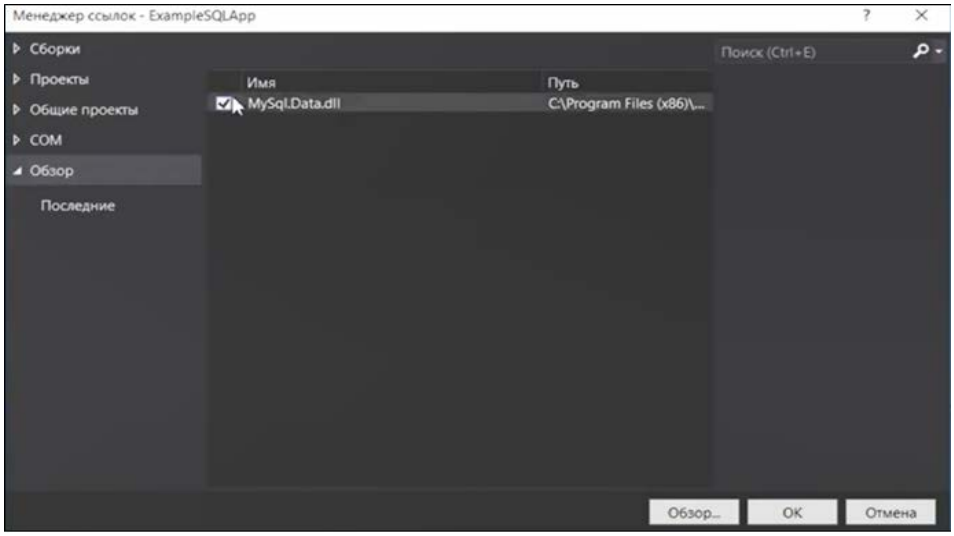


Рисунок 8.2 – Менеджер ссылок

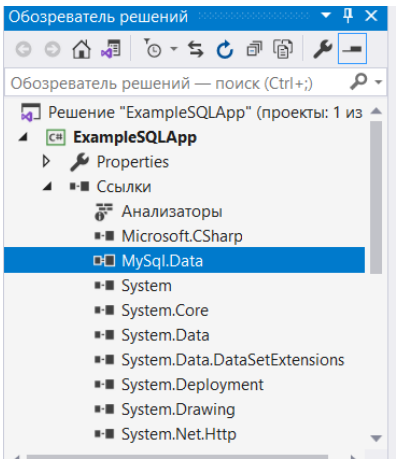


Рисунок 8.3 – Ссылка MySql.Data

Далее становимся на проект и добавляем новый класс DB, описывающий работу с базой данных.

Листинг 8.1 – Код класса DB.cs

```
class DB
{
    //Вводите свой пароль!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!
    //подключение к базе данных
    MySqlConnection connection=new
MySqlConnection("server=localhost;port=3306;username=root;password=1234;databas
e=itproger");

    public void openConnection()
    //метод открывающий базу данных
    {
        if (connection.State == System.Data.ConnectionState.Closed)
            connection.Open();
    }

    public void closeConnection()
    //метод закрывающий базу данных
    {
        if (connection.State == System.Data.ConnectionState.Open)
            connection.Close();
    }

    public MySqlConnection getConnection()
    // возвращает само соединение с базой данных
    {
        return connection;
    }
}
```

Открываем программу MySQL Workbench. Для создания базы данных используется команда `CREATE DATABASE` (рис. 8.4). Она имеет следующий синтаксис:

```
CREATE DATABASE [IF NOT EXISTS] имя_базы_даных;
```

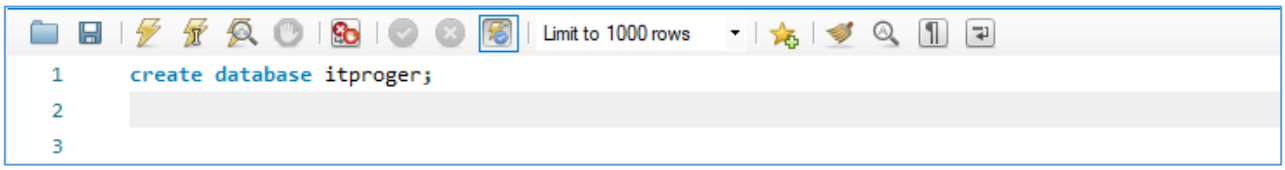


Рисунок 8.4 – Создание базы данных itproger

Удаление базы данных выполняется следующим образом:

```
drop database itproger;
```

После создания БД с ней производятся различные операции: создание таблиц, добавление и получение данных и т. д. Чтобы производить эти операции, надо установить определенную базу данных в качестве используемой. Для этого применяется оператор `USE`.

Для создания таблиц используется команда `CREATE TABLE`. Эта команда применяет ряд операторов, которые определяют столбцы таблицы и их атрибуты. Общий формальный синтаксис команды `CREATE TABLE`:

```
CREATE TABLE название_таблицы
(название_столбца1 тип_данных атрибуты_столбца1,
 название_столбца2 тип_данных атрибуты_столбца2,
 .....
 название_столбцаN тип_данных атрибуты_столбцаN,
 атрибуты_уровня_таблицы
)
```

Рисунок 8.5 – Создание таблицы

После команды `CREATE TABLE` идет название таблицы. Имя таблицы выполняет роль ее идентификатора в базе данных, поэтому оно должно быть уникальным. Затем в скобках перечисляются названия столбцов, их типы данных и атрибуты. В самом конце можно определить атрибуты для всей таблицы. Атрибуты столбцов, а также атрибуты таблицы указывать необязательно.

Создадим таблицу `users`, где будут храниться все пользователи (рис. 8.6). После запуска на выполнение таблица будет создана.

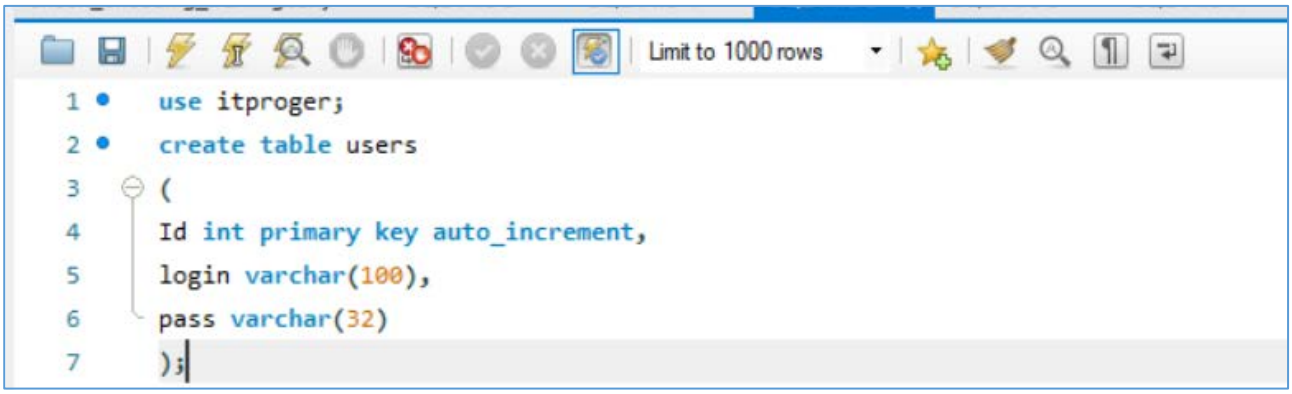


Рисунок 8.6 – Создание таблицы `users`

Что бы посмотреть, как создана таблица следует набрать команды, указанные на рисунке 8.7.

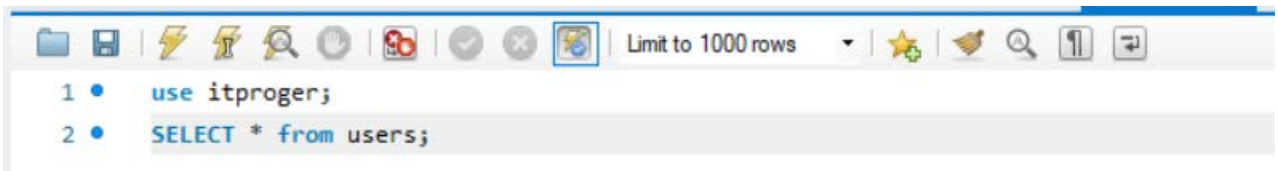


Рисунок 8.7 – Просмотр созданной таблицы

Что бы посмотреть все таблицы, можно ввести команду, представленную на рисунке 8.8.

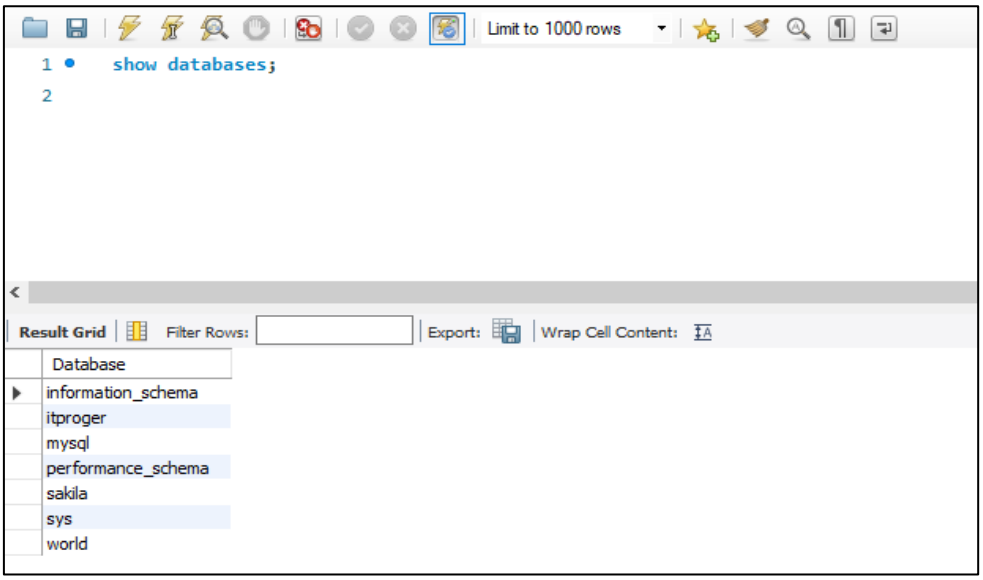


Рисунок 8.8 – Просмотр всех таблиц

***Порядок выполнения работы***

1. Создание окна авторизации

Вначале переименуем форму с именем `Form1` на `LoginForm`. Форма `LoginForm` имеет вид, представленный на рисунке 8.9.

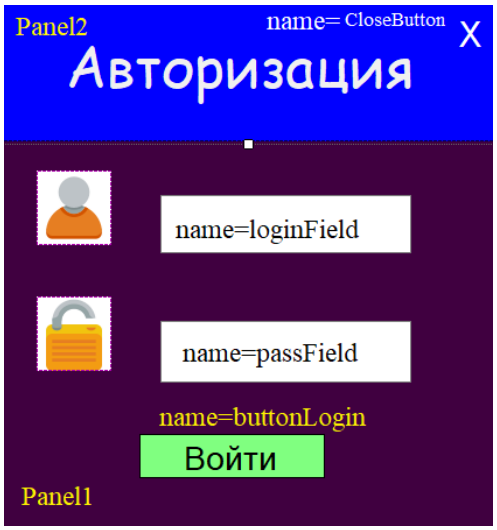


Рисунок 8.9 – Форма LoginForm

На форме нанесены следующие компоненты: `Panel1`, `Panel2`, `pictureBox1`, `pictureBox2`, `TextBox` (`name=loginField`), `TextBox` (`name=passField`), `button` (`name=buttonLogin`).

Для компонента `Panel1` выполним следующие установки свойств: `Dock=Fill` (занимает все пространство), цвет `BackColor` выберите самостоятельно.

Для компонента `Panel2` выполним следующие установки свойств: `Dock=Top` (прижат к верху), цвет `BackColor` так же выберите самостоятельно. Для добавления текстовой надписи воспользуйтесь компонентом `label`

(Text=Авторизация). Подберите размер шрифта (свойство Font) и цвет (свойство ForeColor).

Кнопку закрытия приложения можно реализовать через label (name=CloseButton, Text=X, ForeColor=White, Size=18). Мы также можем изменить свойство Cursor. По умолчанию оно принимает значение Default. Установим значение Hand (курсор руки).

Чтобы удалить системные кнопки, расположенные в правом верхнем углу, нужно перейти на форму и в свойстве FormBorderStyle установить вместо Sizable, выбрать None.

Создадим папку Images (для этого кликаем правой мышью на проект → Добавить → Создать папку). И помещаем в нее изображения.

В pictureBox1, pictureBox2 помещаем картинки как показано на рисунке 8.10. Для картинок установить свойство SizeMode=StretchImage, Size=64;64.

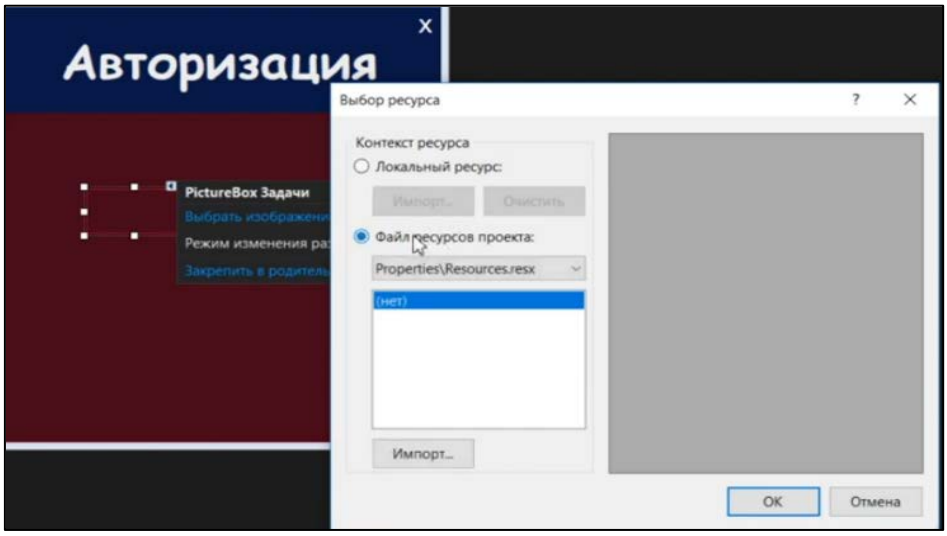


Рисунок 8.10 – Выбор рисунка

Для компонента TextBox (name=loginField) установите свойство MultiLine (рис. 8.11). Для компонента TextBox (name=(name=passField)) свойство MultiLine должно быть отключено.

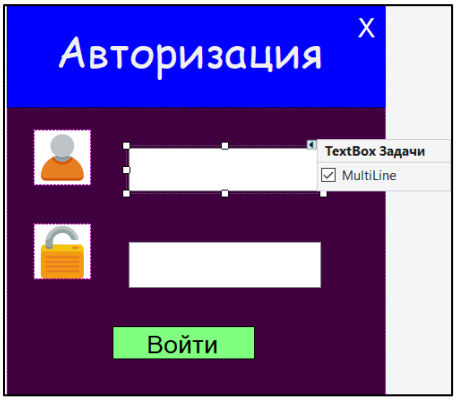


Рисунок 8.11 – Установка свойства MultiLine

Для поля с паролём (`name=passField`) при наборе должны отображаться кружочки. Поэтому для него в свойстве `UseSystemPasswordChar` (использовать системный символ для отображения пароля) установим `True`.

Для кнопки (`name=buttonLogin`) выберем тип кнопки (свойство `FlatStyle=Flat`. Установим так же свойство `FlatAppearance` (рис. 8.12). При нажатии на кнопку цвет будет темный (`MouseDownBackColor`). Свойство `Cursor=Hand`.

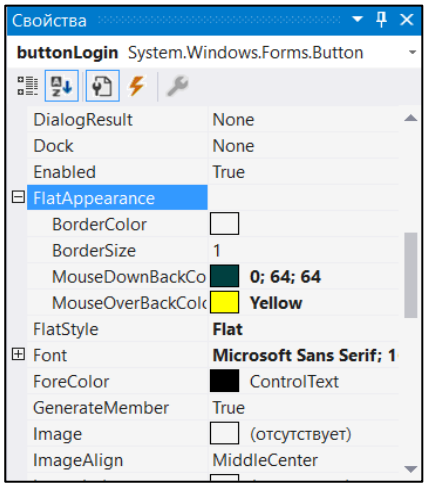


Рисунок 8.12 – Свойство FlatAppearance

Листинг 8.2 – Код LoginForm

```
public partial class LoginForm : Form
{
    public LoginForm()
    {
        InitializeComponent();
        //отключаем автоматический размер
        this.passField.AutoSize = false;
        // 50 потому что у loginField Size 50 (смотрите у себя в программе!!!)
        this.passField.Size = new Size(this.passField.Size.Width, 50);
    }

    private void CloseButton_Click(object sender, EventArgs e)
    {
        //закрытие формы
        this.Close();
    }

    private void CloseButton_MouseEnter(object sender, EventArgs e)
    {
        // событие при наведении мыши на кнопку ее цвет меняется
        CloseButton.ForeColor = Color.Green;
    }

    private void CloseButton_MouseLeave(object sender, EventArgs e)
    {
        // событие когда мышь убираем, меняется цвет
        CloseButton.ForeColor = Color.White;
    }
```

```
    //что бы можно было захватить мышью форму авторизации и ее двигать
    Point lastPoint;//Point-специальный класс для задания координат

    private void panel1_MouseMove(object sender, MouseEventArgs e)
    {
        //событие
        // проверка, если мы нажали на левую кнопку мыши на panel1, то мы можем
двигать форму
        if (e.Button == MouseButtons.Left) {
            this.Left += e.X - lastPoint.X;
            this.Top += e.Y - lastPoint.Y;
        }
    }

    private void panel1_MouseDown(object sender, MouseEventArgs e)
    {
        // событие запоминаем, где нажали мышкой
        lastPoint = new Point(e.X, e.Y);
    }
}
```

2. Авторизация пользователя

При нажатии на кнопку «Войти» на форме `LoginForm` данные, которые введены пользователем в `TextBox` должны проверяться на наличие такого пользователя в базе данных (таблица `users`). Если логин и пароль существуют, то будет происходить авторизация пользователя. Иначе пользователь не авторизуется.

В форме `LoginForm` создаем событие `Click` кнопки `buttonLogin`. Эта функция будет срабатывать каждый раз при нажатии на кнопку. Добавим код из листинга 8.3.

Листинг 8.3 – Код события buttonLogin_Click

```
private void buttonLogin_Click(object sender, EventArgs e)
{
    String loginUser = loginField.Text;
    string passUser = passField.Text;
    // создаем объект класса DB
    DB db = new DB();//объект для подключения к базе данных
    DataTable table = new DataTable();
    MySqlDataAdapter adapter = new MySqlDataAdapter();
    //объект, позволяющий сделать выборку из базы данных
    MySqlCommand command = new MySqlCommand("SELECT * FROM `users` WHERE
`login`=@uL AND `pass`=@uP", db.getConnection());
    command.Parameters.Add("@uL", MySqlDbType.VarChar).Value = loginUser;
    command.Parameters.Add("@uP", MySqlDbType.VarChar).Value = passUser;
    adapter.SelectCommand = command;// через адаптер можем обратиться к команде
которую нужно выполнять
    //данные которые получаем трансформируем в объект table, в котором мы уже
можем посчитать сколько есть элементов
    //сколько записей
    adapter.Fill(table);
    if (table.Rows.Count > 0)//если больше 0, значит пользователь авторизован
        MessageBox.Show("Yes");
```

```
    else
        MessageBox.Show("No");
}
```

Нужно добавить в базу данных пользователя. Для этого входим в MySQLWorkbench и записываем команду, представленную на рисунке 8.13:

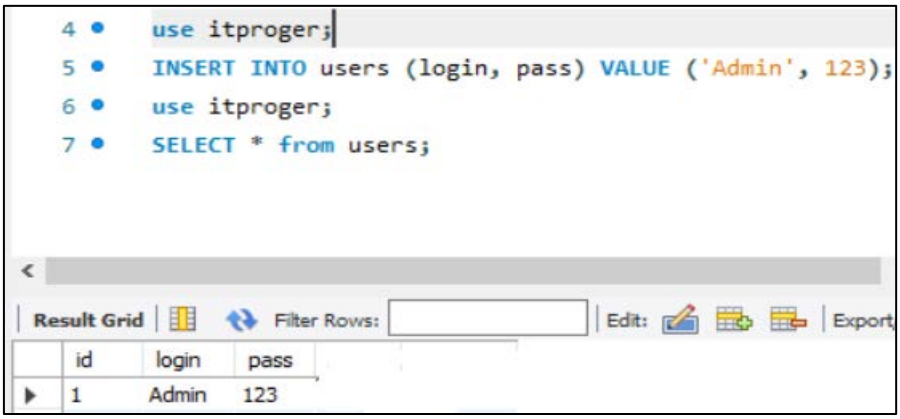


Рисунок 8.13 – Добавление пользователя

Запустите программу и проверьте авторизацию.

3. Создание окна регистрации пользователя.

На проекте кликаем правой мышью и выбираем → Добавить → Форма (Windows Forms) → имя → RegisterForm (рис. 8.14).

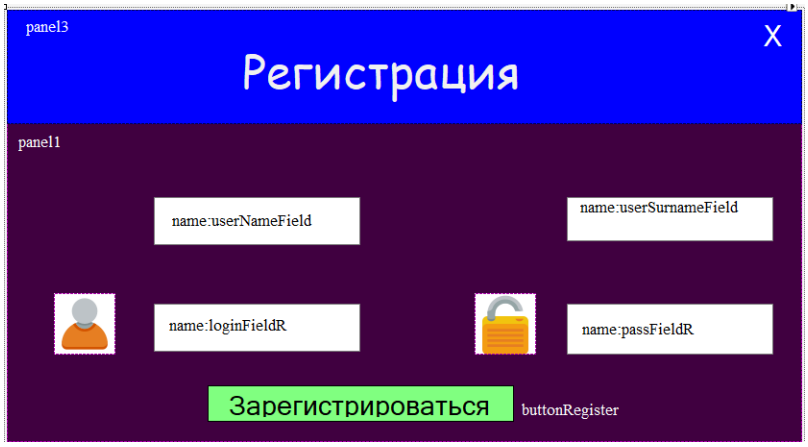


Рисунок 8.14 – Форма регистрации

В Program.cs заменим `new LoginForm()` на new `RegisterForm()`. Задаем свойство формы `FormBorderStyle: None`.

Листинг 8.4 – Код формы RegisterForm

```
public partial class RegisterForm : Form
{
    public RegisterForm()
    {
        InitializeComponent();
```

```
            userNameField.Text = "Введите имя";
            userNameField.ForeColor = Color.Gray;
        }

        private void CloseButtonR_Click(object sender, EventArgs e)
        {
            this.Close();
        }

        private void panel1_MouseDown(object sender, MouseEventArgs e)
        {
            lastPoint = new Point(e.X, e.Y);
        }

        Point lastPoint;

        private void panel1_MouseMove(object sender, MouseEventArgs e)
        {
            if (e.Button == MouseButtons.Left) {
                this.Left += e.X - lastPoint.X;
                this.Top += e.Y - lastPoint.Y;
            }
        }

        private void userNameField_Enter(object sender, EventArgs e)
        {
            //событие срабатывающее при вводе в текстовое поле
            if (userNameField.Text == "Введите имя") {
                //когда пользователь будет вводить надпись пропадает
                userNameField.Text = "";
                userNameField.ForeColor = Color.Black;
            }
        }

        private void userNameField_Leave(object sender, EventArgs e)
        {
            // событие когда выходим из поля
            if (userNameField.Text == "")   //если пользователь ничего не ввел
                userNameField.Text = "Введите имя";
            userNameField.ForeColor = Color.Gray;
        }
    }
```

Добавьте события аналогичные событиям `private void userNameField_Enter (object sender, EventArgs e)` и `private void userNameField_Leave(object sender, EventArgs e)` для оставшихся трех текстовых полей.

4. Регистрация пользователя в базе данных.

Когда пользователь нажмет на кнопку «Зарегистрироваться», должна выполниться проверка, все ли поля заполнены. После удачной проверки эти данные должны поступать в базу данных.

Добавим в таблицу данных два новых столбца (рис. 8.15).

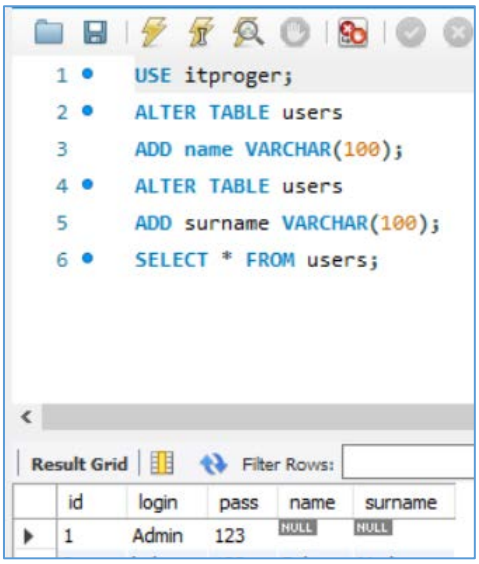


Рисунок 8.15 – Добавление в таблицу `users` новых столбцов

В форме регистрации создаем событие кнопки «Зарегистрироваться» и добавляем код (листинг 8.5).

Листинг 8.5 – Код события buttonRegister_Click

```
private void buttonRegister_Click(object sender, EventArgs e)
{
    if (userNameField.Text == "Введите имя") {
        MessageBox.Show("Введите имя");
        return;
    }
    if (userSurnameField.Text == "") {
        MessageBox.Show("Введите фамилию");
        return;
    }
    //ПРОВЕРЬТЕ ОСТАЛЬНЫЕ ПОЛЯ!!!!! Сделайте то же самое.
    if (isUserExist())
        return;
    DB db = new DB();
    MySqlCommand command = new MySqlCommand("INSERT INTO `users` (`login`,
`pass`, `name`, `surname`) VALUES (@login, @pass, @name, @surname)",
db.getConnection());
    command.Parameters.Add("@login", MySqlDbType.VarChar).Value = loginFieldR.Text;
    command.Parameters.Add("@pass", MySqlDbType.VarChar).Value = passFieldR.Text;
    command.Parameters.Add("@name", MySqlDbType.VarChar).Value = userNameField.Text;
    command.Parameters.Add("@surname", MySqlDbType.VarChar).Value =
userSurnameField.Text;

    //добавление данных в таблицу
    db.openConnection();//открываем соединение с базой
    if (command.ExecuteNonQuery() == 1)// функция выполняет запрос
        MessageBox.Show("Аккаунт был создан");
    else
        MessageBox.Show("Аккаунт не был создан");
    db.closeConnection();//закрываем соединение с базой
}
```

Так же в код формы `RegisterForm` необходимо добавить следующий метод (листинг 8.6).

Листинг 8.6 – Код метода public Boolean isUserExist()

```
public Boolean isUserExist()
{
    //возвращает True если пользователь есть и False если пользователя нет
    DB db = new DB();
    DataTable table = new DataTable();
    MySqlDataAdapter adapter = new MySqlDataAdapter();

    //объект, позволяющий сделать выборку из базы данных
    //выбираем всех пользователей у которых логин такой же как логин который
ввел пользователь в форме регистрации
    MySqlCommand command = new MySqlCommand("SELECT * FROM `users` WHERE
`login`=@uL", db.getConnection());
    command.Parameters.Add("@uL", MySqlDbType.VarChar).Value =
loginFieldR.Text;

    adapter.SelectCommand = command;
    adapter.Fill(table);
    if (table.Rows.Count > 0)// если такой пользователь существует
    {
        MessageBox.Show("Такой логин уже есть, введите другой");
        return true;
    }
    else
        return false;
    }
```

Выполните проверку приложения, введя нового пользователя. Проверьте его наличие в базе данных.

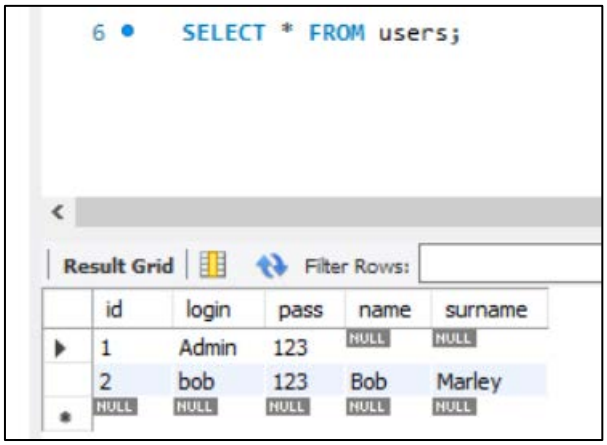


Рисунок 8.16 – Добавление в таблицу users нового пользователя

5. Переход между формами.

Создадим еще одну форму `MainForm`. Задаем свойства: `FormBorderStyle` вместо `Sizable` выбрать `None`, `StartPosition` – `CenterScreen`.

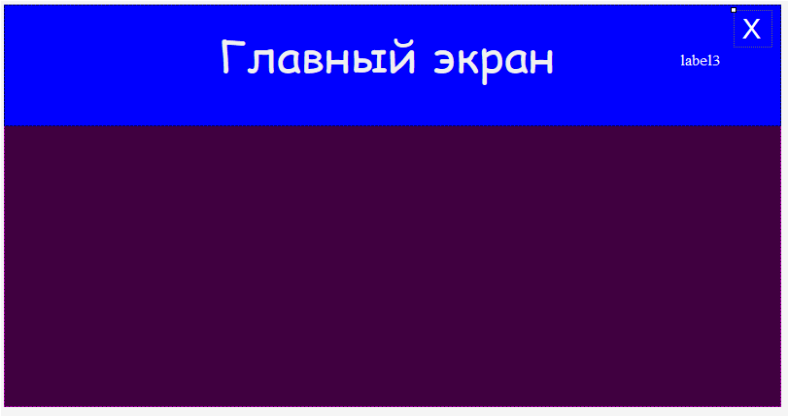


Рисунок 8.17 – MainForm

Листинг 8.7 – Код MainForm

```
public partial class MainForm : Form
{
    public MainForm()
    {
        InitializeComponent();
    }

    private void label3_Click(object sender, EventArgs e)
    {
        Application.Exit();
    }
}
```

В классе Program.cs вернем строку `Application.Run(new LoginForm());`.

В конструкторе `LoginForm` на форме поместим `Label` (`name: registerLable`, `Text: Еще нет аккаунта?`, `ForeColor: White`, размер шрифта `Font:11`.

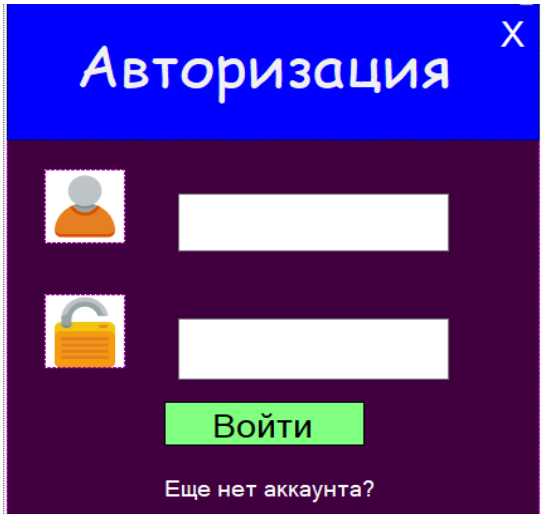


Рисунок 8.18 – Добавление в форму LoginForm нового компонента

При нажатии на этот текст мы должны перейти в окно с регистрацией. Поэтому создаем обработчик события дважды, кликнув по надписи «Еще нет аккаунта?». Добавляем новый код (листинг 8.8).

Листинг 8.8 – Код обработчика событий private void registerLabel_Click

```
private void registerLabel_Click(object sender, EventArgs e)
        {
            this.Hide();//прячем это окно авторизации
            RegisterForm registerForm = new RegisterForm();//потому что на эту
форму хотим перейти
            registerForm.Show();
              }
```

Чтобы правильно программа закрывалась в трех классах (LoginForm, RegisterForm, MainForm) в обработчике событий private void CloseButton_Click (крестик в правом верхнем углу) заменяем строку this.Close(); на Application.Exit();

Перейдем в форму RegisterForm и по аналогии добавим label (Text = Авторизоваться). Создадим обработчика событий для этого компонента и добавим код (листинг 8.9).

Листинг 8.9 – Код обработчика событий private void registerLabel_Click

```
private void registerLabel_Click(object sender, EventArgs e)
{
    //при нажатии мы должны перейти на окно авторизоваться
    this.Hide(); //прячет это окно
    LoginForm loginForm = new LoginForm();
    loginForm.Show();
}
```

В форме `LoginForm` в обработчике событий `private void buttonLogin_Click(object sender, EventArgs e)` заменим

```
if (table.Rows.Count > 0) //если больше 0, значит пользователь авторизован
    MessageBox.Show("Yes");
else
    MessageBox.Show("No");
```

на

```
if (table.Rows.Count > 0) { //если больше 0, значит пользователь авторизован
    this.Hide();//скрываем окно авторизации
    MainForm mainForm = new MainForm();
    mainForm.Show();
}
else
    MessageBox.Show("No");
```

6. Протестируйте работу программы.
7. Добавьте новых пользователей.

8. Проверьте их наличие в таблице, для этого перейдите в MySQL Workbench и сделайте запрос.
9. Оформите главную страницу.

### *Содержание отчета*

Предоставить программу, написанную в Visual Studio на языке C# и результаты ее работы.

### *Контрольные вопросы*

1. Что такое MySQL и для чего он используется?
2. Какие основные типы данных существуют в MySQL?
3. Как создать новую таблицу в MySQL?
4. Как удалить таблицу в MySQL?
5. Как добавить в таблицу данных новые столбцы?

# ЛИТЕРАТУРА

1. Windows Forms C# – уроки создания приложения на itProger / [Электронный ресурс] // Сообщество программистов и онлайн-школа IT профессий itProger. – Режим доступа: https://itproger.com/course/csharp-app. – Дата доступа: 17.04.2024.

2. Боресков, А. В. Основы компьютерной графики : учебник и практикум / А. В. Боресков, Е. В. Шикин ; Московский государственный университет имени М. В. Ломоносова. – Москва : Юрайт, 2020. – 219 с.

3. Дунина, Е. Б. Искусственный интеллект. Лабораторный практикум: учебно-методическое пособие / Е. Б. Дунина, А. С. Соколова, А. А. Корниенко; УО «ВГТУ». – Витебск, 2024. – 88 с.

4. Колошкина, И. Е. Компьютерная графика : учебник и практикум / И. Е. Колошкина, В. А. Селезнев, С. А. Дмитроченко. – 3-е изд., испр. и доп. – Москва : Юрайт, 2021. – 233 с.

5. Культин, Н. Б. Основы программирования в Microsoft Visual C# / Н. Б. Культин. – Санкт-Петербург: БХВ, 2011. – 368 с.

Учебное издание

# РАЗРАБОТКА МУЛЬТИМЕДИЙНЫХ ПРИЛОЖЕНИЙ. МНОГООКОННЫЕ ПРИЛОЖЕНИЯ

Методические указания
по выполнению лабораторных работ

Составители:
Дунина Елена Брониславовна
Соколова Анна Сергеевна
Корниенко Алексей Александрович

Редактор *Р.А. Никифорова*
Корректор *А.С. Прокопюк*
Компьютерная верстка *А.С. Соколова*

---

Подписано к печати 03.09.2025. Усл. печ. листов 3,3.
Уч.-изд. листов 4,4. Заказ № 170.

Учреждение образования «Витебский государственный технологический университет»
210038, г. Витебск, Московский пр., 72.
Отпечатано на ризографе учреждения образования
«Витебский государственный технологический университет».
Свидетельство о государственной регистрации издателя, изготовителя,
распространителя печатных изданий № 1/172 от 12 февраля 2014 г.
Свидетельство о государственной регистрации издателя, изготовителя,
распространителя печатных изданий № 3/1497 от 30 мая 2017 г.