

ОСНОВЫ ПРОГРАММИРОВАНИЯ В СРЕДЕ C#

1 Основы языка C#	1
1.1 Примеры программ на C#	1
1.1.1 Составление линейных программ	1
1.1.2 Составление циклических программ	2
2 Работа с массивами в языке C#	3
2.1 Примеры программ для работы с массивами в C#	3
2.1.1 Одномерные массивы	3
2.1.2 Двумерные «прямоугольные» массивы	4
2.1.3 Двумерные «ступенчатые» массивы	9
3 СТРОКИ И РЕГУЛЯРНЫЕ ВЫРАЖЕНИЯ	10
3.1 Примеры программ для работы со строками	10
3.1.1 Пример программы для работы с классом String	10
3.1.2 Пример работы с классом StringBuilder	11
3.1.3 Пример работы с регулярными выражениями	12
ПРИЛОЖЕНИЕ А. Справочные данные для лабораторной работы № 1	13
ПРИЛОЖЕНИЕ Б. Справочные данные для лабораторной работы № 2	14
ПРИЛОЖЕНИЕ В. Справочные данные для лабораторной работы № 3	14

1 Основы языка C#

1.1 Примеры программ на C#

1.1.1 Составление линейных программ

Составление линейных программ рассмотрим на следующем примере:

Для известной переменной «А» вычислить

$$Z(A) = \cos(A) + \sin(A) + \cos(3A) + \sin(3A);$$

Для решения данной задачи создайте форму, изображенную на рисунке 1.1.

Рисунок 1.1 – Форма для вычисления $Z(A)$

Для события Click кнопки button1 запишите следующий программный код:

```
private void button1_Click(object sender, EventArgs e)
{
    double a = Convert.ToDouble(textBox1.Text);
    double z = Math.Cos(a) + Math.Sin(a) + Math.Cos(3 * a) +
    Math.Sin(3 * a);
    textBox2.Text = Convert.ToString(z);
}
```

Проверьте работу приложения.

1.1.2 Составление циклических программ

Составление циклических программ рассмотрим на следующем примере: Найти сумму цифр натурального числа.

Для решения данной задачи создайте форму, изображенную на рисунке 1.2.

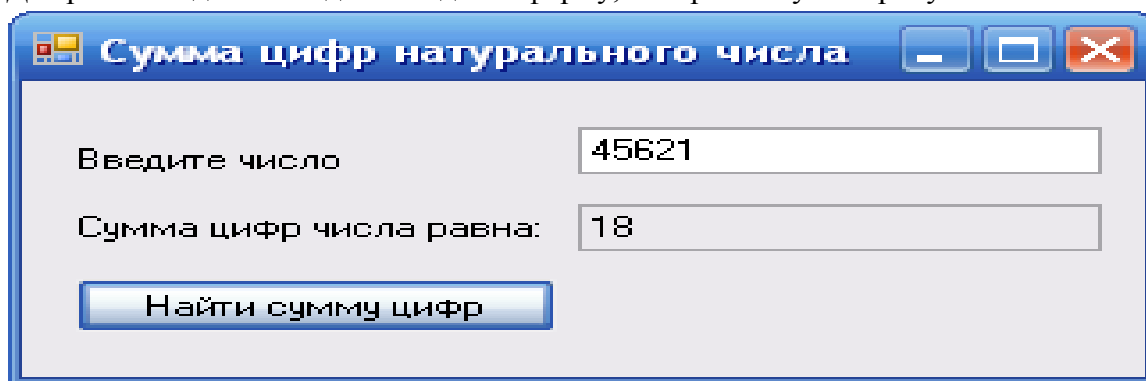


Рисунок 1.2 – Форма для вычисления суммы цифр натурального числа

Исходное число вводится в поле textBox1.

Результат выводится в поле textBox2, имеющее свойство ReadOnly:=True.

Расчет выполняется при нажатии командной кнопки button1.

Для события Click кнопки button1 запишите следующий программный код:

```
private void button1_Click(object sender, EventArgs e)
{
    int n = Convert.ToInt32(textBox1.Text);
    int s = 0;
    while (n!=0)
    {
        s = s + n % 10;
        n = n / 10;
    }
    textBox2.Text = Convert.ToString(s);
}
```

Для корректной работы программы в окно textBox1 требуется вводить только цифры от 0 до 9. Для проверки вводимых символов запрограммируем событие KeyPress окна ввода textBox1 следующим образом:

```
private void textBox1_KeyPress(object sender, KeyPressEventArgs e)
{
    if (!(e.KeyChar >= '0' && e.KeyChar <= '9'))
```

```

        e.KeyChar = '\0';
    }
}

```

2 Работа с массивами в языке C#

2.1 Примеры программ для работы с массивами в C#

2.1.1 Одномерные массивы

Работу с одномерными массивами рассмотрим на следующем примере:
 В одномерном массиве, состоящем из n вещественных элементов, вычислить сумму элементов массива, расположенных до минимального элемента.
 Для решения данной задачи создайте форму, изображенную на рисунке 2.1.

The screenshot shows a Windows application window titled "Form1". Inside the window, there is a label "Количество элементов" followed by a text box containing the number "5". Below this is a list box containing the numbers 6, 3, 9, 1, and 4. To the right of the list box are three buttons: "Очистить", "Заполнить", and "Сумма элементов массива до минимального". At the bottom of the window, there is a label "Сумма элементов массива до минимального" followed by a text box containing the number "18".

Рисунок 2.1 – Форма для работы с одномерным массивом

Список используемых элементов управления приведен в таблице 2.1.

Таблица 2.1 - Список используемых элементов управления

Элемент управления	Класс	Описание
Button1	Button	Командная кнопка «Очистить»
Button2	Button	Командная кнопка «Заполнить»
Button3	Button	Командная кнопка «Сумма элементов массива до минимального»
Label1	Label	Метка «Количество элементов»
Label2	Label	Метка «Сумма элементов массива до минимального»
textBox1	textBox	Окно ввода количества элементов в массиве
textBox2	textBox	Многострочное окно вывода массива
textBox3	textBox	Окно вывода суммы элементов массива до минимального

Последовательность действий:

1. Установите у элемента управления textBox2 свойство Multiline=true для вывода каждого элемента массива в отдельной строке
2. Установите у элемента управления textBox2 свойство ScrollBars=Vertical для обеспечения скроллинга.
3. Объявите следующие глобальные переменные:

```
int[] a ;  
int n;
```

4. Для события Click кнопки Button1 запишите следующий программный код:

```
private void button1_Click(object sender, EventArgs e)  
{  
    textBox2.Clear();  
}
```

5. Для события Click кнопки Button2 запишите следующий программный код:

```
private void button2_Click(object sender, EventArgs e)  
{  
    n = Convert.ToInt32(textBox1.Text);  
    a = new int[n];  
    Random b = new Random();  
    for (int i = 0; i < n; ++i)  
    {  
        a[i] = b.Next(10);  
  
        textBox2.Text+=Convert.ToString(a[i])+(char)13+  
            (char)10;  
    }  
}
```

6. Для события Click кнопки Button3 запишите следующий программный код:

```
private void button3_Click(object sender, EventArgs e)  
{  
    int sum = 0;  
    int min = a[0];  
    int minind=0;  
    for (int i = 1; i < n; ++i)  
    {  
        if (a[i] < min)  
        {  
            min = a[i];  
            minind = i;  
        }  
    }  
    for (int i = 0; i < minind; ++i)  
        sum += a[i];  
    textBox3.Text = Convert.ToString(sum);  
}
```

7. Проверьте работу приложения

2.1.2 Двумерные «прямоугольные» массивы

Работу с двумерными «прямоугольными» массивами рассмотрим на примере задачи умножения прямоугольных матриц $C=A*B$.

Для решения данной задачи будем использовать элемент управления DataGridView.

Элемент управления DataGridView позволяет отображать таблицы. Главное назначение DataGridView - связывание с таблицами внешних источников данных, прежде всего с

таблицами баз данных. Но данный элемент имеет и другое применение - ввод и отображение матриц - двумерных массивов.

На рисунке 2.2 показан возможный вид формы, поддерживающей работу пользователя. Форма показана в тот момент, когда пользователь уже задал размеры и значения исходных матриц, выполнил умножение матриц и получил результат.

Список используемых элементов управления приведен в таблице 2.2.

1. Задайте поля, определяющие размеры матриц и сами матрицы:

```
int m, n, p;           //размеры матриц
```

```
double[,] A, B, C;     //сами матрицы
```

Таблица 2.2 - Список используемых элементов управления

Рисунок 2.2 - Приложение, для перемножения матриц

Элемент управления	Класс	Описание
label1	label	Надпись «матрица А»
label2	label	Надпись «матрица В»
label3	label	Надпись «матрица С»
label4	label	Надпись «m»
label5	label	Надпись «n»
label6	label	Надпись «p»
label7	label	Невидимая надпись для вывода сообщения вида «Значение элемента A[i,j] не корректно. Повторите ввод»
label8	label	Невидимая надпись для вывода сообщения вида «Значение элемента B[i,j] не корректно. Повторите ввод»
textBox1	TextBox	Окно ввода m
textBox2	TextBox	Окно ввода n
textBox3	TextBox	Окно ввода p
dataGridView1	DataGridView	Матрица А
dataGridView2	DataGridView	Матрица В
dataGridView3	DataGridView	Матрица С
button1	Button	Командная кнопка «Создание матриц»
button2	Button	Командная кнопка «Проверка корректности ввода»
button3	Button	Командная кнопка «Перемножение матриц»

2. Рассмотрим теперь, как выглядит обработчик события «Click» командной кнопки «Создание матриц». Предполагается, что пользователь разумен и, прежде чем нажать эту кнопку, задает размеры матриц в соответствующих текстовых окнах. При перемножении матриц размеры матриц должны быть согласованы - число столбцов первого сомножителя должно совпадать с числом строк второго сомножителя, а размеры результирующей матрицы определяются размерами сомножителей. Поэтому для трех матриц в данном случае достаточно задать не шесть, а три параметра, определяющих размеры. Обработчик события выполняет три задачи - создает сами матрицы, осуществляет чистку элементов управления DataGridView, удаляя предыдущее состояние, затем добавляет столбцы и строки в эти элементы в полном соответствии с заданными размерами матриц. Программный код имеет следующий вид::

```
private void button1_Click(object sender, EventArgs e)
{
    m = Convert.ToInt32(textBox1.Text);
    n = Convert.ToInt32(textBox2.Text);
    p = Convert.ToInt32(textBox3.Text);
    A = new double[m, n];
    B = new double[n, p];
    C = new double[m, p];
    // Чистка DataGridView, если они не пусты
    int k = 0;
    k = dataGridView1.ColumnCount;
    if (k != 0)
        for (int i = 0; i < k; i++)
            dataGridView1.Columns.RemoveAt(0);
    dataGridView2.Columns.Clear();
    dataGridView3.Columns.Clear();
    // Заполнение DataGridView столбцами
    AddColumns(n, dataGridView1);
    AddColumns(p, dataGridView2);
    AddColumns(p, dataGridView3);
    // Заполнение DataGridView строками
    AddRows(m, dataGridView1);
    AddRows(n, dataGridView2);
    AddRows(m, dataGridView3);
}
```

Чистка предыдущего состояния элементов DataGridView сводится к удалению столбцов. В программе показаны два возможных способа выполнения этой операции. Для первого элемента показано, как можно работать с коллекцией столбцов. Организуется цикл по числу столбцов коллекции и в цикле выполняется метод RemoveAt, аргументом которого является индекс удаляемого столбца. Поскольку после удаления столбца происходит перенумерация столбцов, то на каждом шаге цикла удаляется первый столбец, индекс которого всегда равен нулю. Удаление столбцов коллекции можно выполнить сразу - вызывая метод Clear() коллекции, что и делается для остальных двух элементов DataGridView.

После чистки предыдущего состояния, можно задать новую конфигурацию элемента, добавив в него вначале нужное количество столбцов, а затем и строк. Эти задачи выполняют специально написанные процедуры AddColumns и AddRows.

3. Текст процедуры AddColumns:

```
private void AddColumns(int n, DataGridView dgw)
{
    // добавляет n столбцов в элемент управления dgw
    DataGridViewColumn column;
```

```

for (int i = 0; i < n; i++)
{
    column = new DataGridViewTextBoxColumn();
    column.DataPropertyName = "Column" + i.ToString();
    column.Name = "Column" + i.ToString();
    dgw.Columns.Add(column);
}
}

```

В процедуре создаются столбцы в коллекции Columns по одному. В цикле по числу столбцов матрицы, которую должен отображать элемент управления DataGridView, вызывается метод Add этой коллекции, создающий очередной столбец. Одновременно в этом же цикле создается и имя столбца (свойство Name), отображаемое в форме. Показана возможность формирования еще одного имени (DataPropertyName), используемого при связывании со столбцом таблицы внешнего источника данных. В данном примере это имя не используется.

4. Текст процедуры AddRows:

```

private void AddRows(int m, DataGridView dgw)
{
    // добавляет m строк в элемент управления dgw
    for (int i = 0; i < m; i++)
    {
        dgw.Rows.Add();
        dgw.Rows[i].HeaderCell.Value = "row" + i.ToString();
    }
}

```

Создав столбцы, нужно создать еще и нужное количество строк у каждого из элементов DataGridView. Делается это аналогичным образом, вызывая метод Add коллекции Rows. Несколько по-другому задаются имена строк - для этого используется специальный объект HeaderCell, имеющийся у каждой строки и задающий ячейку заголовка.

5. После того как сформированы строки и столбцы, элемент DataGridView готов к тому, чтобы пользователь или программа вводила значения в ячейки сформированной таблицы.

6. Обработчик события «Click» командной кнопки «Проверка корректности ввода» имеет вид:

```

private void button2_Click(object sender, EventArgs e)
{
    string elem = "";
    bool correct = true;
    for (int i = 0; i < m; i++)
        for (int j = 0; j < n; j++)
        {
            try
            {
                elem = dataGridView1.Rows[i].Cells[j].Value.ToString();
                A[i, j] = Convert.ToDouble(elem);
                label7.Text = "";
            }
            catch (Exception any)
            {
                label7.Text = "Значение элемента A[" + i.ToString() + " , " + j.ToString() + " ] не корректно. Повторите ввод";
                dataGridView1.Rows[i].Cells[j].Selected = true;
            }
        }
}

```

```

        return;
    }
}
for (int i = 0; i < m; i++)
    for (int j = 0; j < p; j++)
    {
        try
        {
            elem =
dataGridView2.Rows[i].Cells[j].Value.ToString();
            B[i, j] = Convert.ToDouble(elem);
            label8.Text = "";
        }
        catch (Exception any)
        {
            label8.Text = "Значение элемента B[" +
i.ToString() + " , " + j.ToString() + " ] не корректно. Повторите
ВВОД";
dataGridView2.Rows[i].Cells[j].Selected = true;
            return;
        }
    }
}

```

Предполагается, что пользователь разумен и, прежде чем нажать эту кнопку, задает значения элементов перемножаемых матриц в соответствующих ячейках подготовленных таблиц первых двух элементов DataGridView. Обработчик события выполняет следующие задачи - в цикле читает элементы, записанные пользователем в таблицы DataGridView, проверяет их корректность и в случае успеха переписывает их в матрицы.

Основная задача переноса данных из таблицы элемента DataGridView в соответствующий массив не вызывает проблем. Конструкция Rows[i].Cells[j] позволяет добраться до нужного элемента таблицы, после чего остается присвоить его значение элементу массива.

При вводе основной проблемой является обеспечение корректности вводимых данных. Проверка корректности выполняется, после того как пользователь полностью заполнил таблицы, при этом некоторые элементы он мог задать некорректно. Просматривая таблицу, необходимо обнаружить некорректно заданные значения и предоставить возможность их исправления. В программе предлагается следующее решение этой проблемы: преобразование данных, введенных пользователем, в значение, допустимое для элементов матрицы A, помещается в охраняемый блок. Если данные некорректны и возникает исключительная ситуация, то она перехватывается универсальным обработчиком catch(Exception).

В данном варианте нет цикла, работающего до тех пор, пока не будет введено корректное значение. Обработчик исключения просто прерывает работу по переносу данных, вызывая оператор return. Но предварительно он формирует информационное сообщение об ошибке и выводит его в форму. (Специально для этих целей у формы были заготовлены две метки).

В сообщении пользователю предлагается исправить некорректно заданный элемент и повторить ввод - повторно нажать командную кнопку «перенести данные в массив».

7. Обработчик события «Click» командной кнопки «Перемножение матриц» реализует умножение матриц и отображает полученный результат в таблице соответствующего элемента

DataGridView:

```
private void button3_Click(object sender, EventArgs e)
{
    MulMatr(A, B, C);
    FillDG();
}
```

8. Процедура перемножения матриц MulMatr имеет вид:

```
void MulMatr(double[,] A, double[,] B, double[,] C)
{
    int m = A.GetLength(0);
    int n = A.GetLength(1);
    int p = B.GetLength(1);
    double s = 0;
    for(int i=0; i<m;i++)
        for (int j = 0; j < p; j++)
        {
            s = 0;
            for (int k = 0; k < n; k++)
                s += A[i, k] * B[k, j];
            C[i, j] = s;
        }
}
```

9. Процедура вывода результата перемножения в элемент управления dataGridView3 имеет вид:

```
void FillDG()
{
    for (int i = 0; i < m; i++)
        for (int j = 0; j < p; j++)
            dataGridView3.Rows[i].Cells[j].Value = C[i,
j].ToString();
}
```

10. Проверьте работу приложения

ПРИМЕЧАНИЕ. Приложения для работы с двумерными массивами можно выполнить как консольное приложение.

2.1.3 Двумерные «ступенчатые» массивы

Для демонстрации работы со «ступенчатыми» массивами рассмотрим консольное приложение и будем использовать класс Array. Приложение формирует двумерный массив, состоящий из трех строк, строки имеют различное количество столбцов.

Листинг приложения имеет вид:

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;

namespace stmas
{
    class Program
    {
        static void Main(string[] args)
        {
            int[][] a=new int [3][];
```

```

a[0]=new int [5]{24,50,18,3,16};
a[1]=new int [3]{7,9,-1};
a[2]=new int [4]{6,15,3,1};
Console.WriteLine("Исходный массив:");
for (int i=0;i<a.Length;++i)
{
    for (int j=0;j<a[i].Length;++j)
        Console.Write("\t"+a[i][j]);
    Console.WriteLine();
}
Console.WriteLine(Array.IndexOf(a[0],18));
}
}

```

Результат работы приложения приведен на рисунке 2.3.

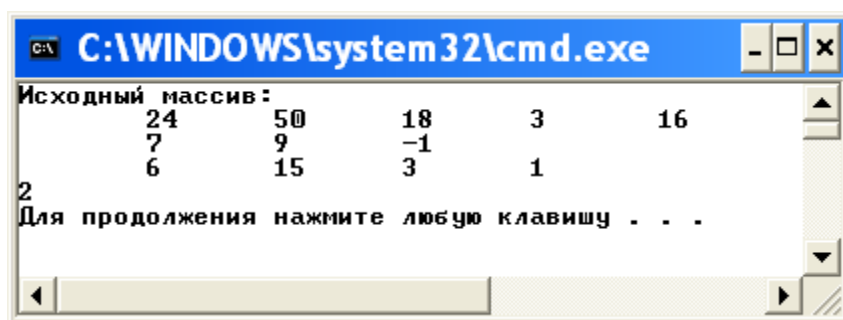


Рисунок 2.3 – Пример работы со «ступенчатыми массивами»

3 СТРОКИ И РЕГУЛЯРНЫЕ ВЫРАЖЕНИЯ

3.1 Примеры программ для работы со строками

3.1.1 Пример программы для работы с классом String

Рассмотрим пример программы шифрования методом Цезаря (модификация метода). Исходная строка модифицируется следующим образом – каждый символ заменяется на другой с ASCII-кодом, следующим в алфавитном порядке.

Последовательность действий:

1. Создайте приложение оконное приложение, изображенное на рисунке 3.1

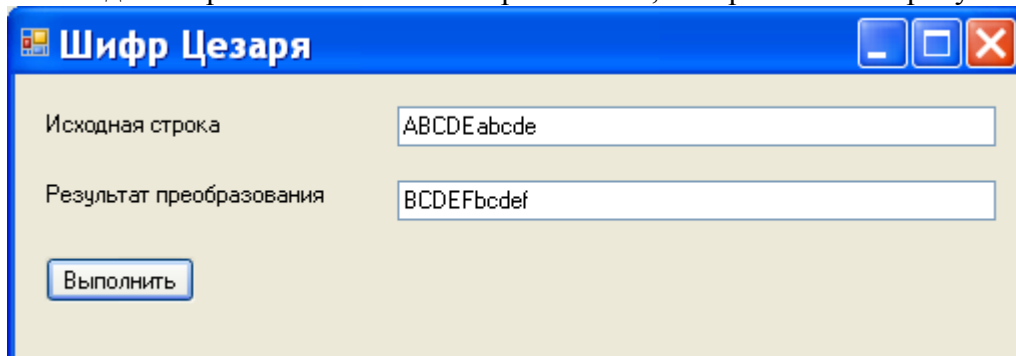


Рисунок 3.1 – Пример программы шифрования по методу Цезаря

Список используемых элементов управления приведен в таблице 3.1.

Таблица 3.1 - Список используемых элементов управления

Элемент управления	Класс	Описание
Label1	Label	Метка «Исходная строка»
Label2	Label	Метка «Результат преобразования»
textBox1	TextBox1	Окно ввода исходной строки
textBox2	TextBox2	Окно вывода результата
Button1	Button	Командная кнопка «Выполнить»

2. Для события Click кнопки Button1 запишите следующий программный код:

```
private void button1_Click(object sender, EventArgs e)
{
    string s = textBox1.Text;
    for (int i='z';i>=(int)'a';i--)
    {
        char old1=(char) i;
        char new1=(char) (i+1);
        s=s.Replace(old1,new1);
    }
    for (int i='Z';i>=(int)'A';i--)
    {
        char old1=(char) i;
        char new1=(char) (i+1);
        s=s.Replace(old1,new1);
    }
    textBox2.Text=s;
}
```

Класс String весьма мощный класс, но он является неизменяемым (о чем говорилось в разделе 1). То есть если однажды строковый объект инициализирован, он уже не может быть изменен. Методы и операции, модифицирующие содержимое строк, на самом деле создают новые строки, копируя при необходимости старые. В данном примере метод Replace() при вызове каждый раз создает новую строку. Следовательно, в результате такого процесса шифрования появляются строковые объекты, которые будут храниться в куче до тех пор пока сборщик мусора не уничтожит их.

Для того, чтобы справиться с этой проблемой и предусмотрен класс StringBuilder.

3.1.2 Пример работы с классом StringBuilder

Работу с классом StringBuilder рассмотрим на предыдущем примере. Оконное приложение изображено на рисунке 3.1, а список элементов управления приведен в таблице 3.1.

Программный код для события Click кнопки Button1 имеет следующий вид:

```
private void button1_Click(object sender, EventArgs e)
{
    StringBuilder s = new StringBuilder(textBox1.Text);
    for (int i = 'z'; i >= (int)'a'; i--)
    {
        char old1 = (char)i;
        char new1 = (char)(i + 1);
        s = s.Replace(old1, new1);
    }
    for (int i = 'Z'; i >= (int)'A'; i--)
    {
        char old1 = (char)i;
```

```

        char new1 = (char)(i + 1);
        s = s.Replace(old1, new1);
    }
    textBox2.Text = s.ToString();
}

```

В данном программном фрагменте используется метод `StringBuilder.Replace()`, который выполняет ту же работу, что и `String.Replace()`, но без копирования строки.

3.1.3 Пример работы с регулярными выражениями

Рассмотрим пример программы, выполняющей подсчет количества слов в текстовом файле.

Последовательность действий:

3. Создайте приложение оконное приложение, изображенное на рисунке 3.2.

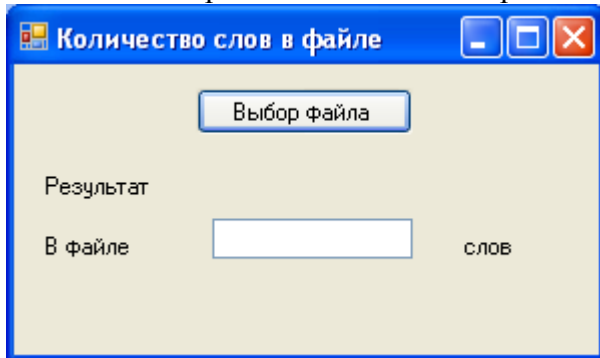


Рисунок 3.2– Пример приложения

Список используемых элементов управления приведен в таблице 3.2.

Таблица 3.2 - Список используемых элементов управления

Элемент управления	Класс	Описание
Button1	Button	Командная кнопка «Выбор файла»
Label1	Label	Метка «Результат»
Label2	Label	Метка «слов»
Label3	Label	Метка «в файле»
openFileDialog1	openFileDialog	Невидимый элемент управления для выбора текстового файла

4. Для работы с файлами и регулярными выражениями добавьте пространство имен `using System.IO` и `using System.Text.RegularExpressions`;
5. Для события `Click` кнопки `Button1` запишите следующий программный код:

```

private void button1_Click(object sender, EventArgs e)
{
    // Диалог выбора файла
    if(openFileDialog1.ShowDialog() == DialogResult.OK)
    {
        // Открытие выбранного файла
        StreamReader f = new StreamReader(openFileDialog1.FileName);
        String pattern = @"\b(w+)";
        String s;
        int total = 0;
        Regex r = new Regex(pattern);
        // Чтение выбранного файла по строкам
        while ((s = f.ReadLine()) != null)

```

```

    {
        Match m = r.Match(s);
// Поиск регулярного выражения в очередной строке
        while (m.Success)
        {
            total++;
            m = m.NextMatch();
        }
    }
    f.Close();
    textBox1.Text = Convert.ToString(total);
}
}

```

ПРИЛОЖЕНИЕ А. Справочные данные для лабораторной работы № 1

Таблица А.1 Основные типы данных C#

Тип C#	Тип CLR	Размер в байтах	Пояснение
int	Int32	4	Целое (со знаком)
float	Single	4	Вещественное число
char	Char	-	Символ (Unicode)
bool	Boolean	-	Логический тип
short	Int16	2	Короткое целое (со знаком)
long	Int64	8	Длинное целое (со знаком)
string	String	-	строка
byte	Byte	1	байт
decimal	Decimal	8	Вещественное число фиксированной точности

Таблица А.2 – Арифметические операторы C#

Оператор	Значение
+	Сложение
-	Вычитание (также унарный минус)
*	Умножение
/	Деление
%	Взятие по модулю (остаток от деления)
++	Инкремент
--	Декремент

Таблица А.3 – Поразрядные операции C#

Операция	Назначение	Пример
&	Поразрядное И	i & 16
	Поразрядное ИЛИ	i 12
^	Поразрядное Исключающее ИЛИ	k ^ 10
~	Поразрядное НЕ	~a
<<	Поразрядный сдвиг влево	<<2
>>	Поразрядный сдвиг вправо	>>2

Таблица А.4 - Логические операторы C#

Оператор	Описание	Пример
&&	Логическое И. Результат равен true, только если	(x==8) && (y==5)

	оба операнда равны true	
	Логическое ИЛИ. Результат равен false, только если оба операнда равны false	(y>8) (y<5)
!	Отрицание. Изменяет логическое значение на противоположное	if(!(a==b))...

ПРИЛОЖЕНИЕ Б. Справочные данные для лабораторной работы № 2

Таблица Б.1- Методы и свойства класса System.Array

Элемент	Вид	Описание
Length	Свойство	Количество элементов массива (по всем размерностям)
Rank	Свойство	Количество размерностей массива
BinarySearch	Статический метод	Двоичный поиск в отсортированном массиве
Clear	Статический метод	Присваивание элементам массива значений по умолчанию
Copy	Статический метод	Копирование заданного диапазона элементов одного массива в другой массив
CopyTo	Метод	Копирование всех элементов текущего одномерного массива в другой одномерный массив
GetValue	Метод	Получение значения элемента массива
IndexOf	Статический метод	Поиск первого вхождения элемента в одномерный массив
LastIndexOf	Статический метод	Поиск последнего вхождения элемента в одномерный массив
Reverse	Статический метод	Изменение порядка следования элементов на обратный
SetValue	Метод	Установка значения элемента массива
Sort	Статический метод	Упорядочение элементов одномерного массива

ПРИЛОЖЕНИЕ В. Справочные данные для лабораторной работы № 3

Таблица В.1 – Регулярные выражения. Классы символов

Класс символов	Описание	Пример
.	Любой символ, кроме \n	Выражение c.t соответствует фрагментам cat, cut, clt, c{t и т. д.
[]	Любой одиночный символ из последовательности, записанной внутри скобок. Допускается использование диапазонов символов	Выражение c[aul]t соответствует фрагментам cat, cut и clt, а выражение c[a-z]t — фрагментам cat, cbt, cct, celt,..., czt
[^]	Любой одиночный символ, не входящий в последовательность, записанную внутри скобок. Допускается использование диапазонов символов	Выражение c[^aul]t соответствует фрагментам cbt, c2t, cXt и т. д., а выражение c[^a-zA-Z]t — фрагментам cit, clt, c4t, c3t и т. д.
\w	Любой алфавитно-цифровой символ, то есть символ из множества прописных и строчных букв и десятичных цифр	Выражение c\wt соответствует фрагментам cat, cut, clt, cЮt и т. д., но не соответствует фрагментам c{t, c;t и т. д.

\W	Любой <i>не</i> алфавитно-цифровой символ, то есть символ, не входящий в множество прописных и строчных букв и десятичных цифр	Выражение <code>c\Wt</code> соответствует фрагментам <code>c{t, c;t, c t</code> и т. д., но не соответствует фрагментам <code>cat, cut, clt, cЮt</code> и т. д.
\s	Любой пробельный символ, например символ пробела, табуляции (<code>\t</code> , <code>\v</code>), перевода строки (<code>\n</code> , <code>\r</code>), новой страницы (<code>\f</code>)	Выражение <code>\s\w\w\w\s</code> соответствует любому слову из трех букв, окруженному пробельными символами
\S	Любой <i>не</i> пробельный символ, то есть символ, не входящий в множество пробельных	Выражение <code>\s\S\S\s</code> соответствует любым двум непробельным символам, окруженным пробельными

Класс символов	Описание	Пример
\d	Любая десятичная цифра	Выражение c\dт соответствует фрагментам c1t, c2t,..., c9t
\D	Любой символ, не являющийся десятичной цифрой	Выражение c\Dт не соответствует фрагментам c1t, c2t, ..., c9t

Таблица В.2- Регулярные выражения. Уточняющие метасимволы

Метасимвол	Описание
^	Фрагмент, совпадающий с регулярным выражением, следует искать только в начале строки
\$	Фрагмент, совпадающий с регулярным выражением, следует искать только в конце строки
\A	Фрагмент, совпадающий с регулярным выражением, следует искать только в начале многострочной строки
\Z	Фрагмент, совпадающий с регулярным выражением, следует искать только в конце многострочной строки
\b	Фрагмент, совпадающий с регулярным выражением, начинается или заканчивается на границе слова (то есть между символами, соответствующими метасимволам \w и \W)
\B	Фрагмент, совпадающий с регулярным выражением, не должен встречаться на границе слова

Таблица В.3 – Регулярные выражения. Повторители

Метасимвол	Описание	Пример
*	Ноль или более повторений предыдущего элемента	Выражение ca*t соответствует фрагментам ct, cat, caat, caaaaaaaaaaat и т.д.
+	Одно или более повторений предыдущего элемента	Выражение ca+t соответствует фрагментам cat, caat, caaaaaaaaaaat и т.д.
?	Ни одного или одно повторение предыдущего элемента	Выражение ca?t соответствует фрагментам ct и cat
{n}	Ровно n повторений предыдущего элемента	Выражение ca{3}t соответствует фрагменту caaat, а выражение (cat){2} – фрагменту catcat (круглые скобки служат для группировки символов)
{n.}	По крайней мере n повторений предыдущего элемента	Выражение ca{3.}t соответствует фрагментам caaat, caaaat, caaaaaaaaaaat и т.д.
{n.m}	От n до m повторений предыдущего элемента	Выражение ca{2.4}t соответствует фрагментам caat, caaat и caaaat

Таблица Г.6 - Наиболее важные элементы базового класса TextWriter

Элемент	Описание
Close	Заккрыть файл и освободить связанные с ним ресурсы. Если в процессе записи используется буфер, он будет автоматически очищен
Flush	Очистить все буферы для текущего файла и записать накопленные в них данные в место их постоянного хранения. Сам файл при этом не закрывается
NewLine	Используется для задания последовательности символов, означающих начало новой строки. По умолчанию используется последовательность «возврат каретки» — «перевод строки» (\r\n)
Write	Записать фрагмент текста в поток
WriteLine	Записать строку в поток и перейти на другую строку

Таблица Г.7 - Наиболее важные элементы класса TextReader

Элемент	Описание
Peek	Возвратить следующий символ, не изменяя позицию указателя в файле
Read	Считать данные из входного потока
ReadBlock	Считать из входного потока указанное пользователем количество символов и записать их в буфер, начиная с заданной позиции
ReadLine	Считать строку из текущего потока и вернуть ее как значение типа string. Пустая строка null означает конец файла (EOF)
ReadToEnd	Считать все символы до конца потока, начиная с текущей позиции, и вернуть считанные данные как одну строку типа string

Таблица Г.8 - Наиболее важные элементы класса BinaryWriter

Элемент	Описание
BaseStream	Базовый поток, с которым работает объект BinaryWriter
Close	Заккрыть поток
Flush	Очистить буфер
Seek	Установить позицию в текущем потоке
Write	Записать значение в текущий поток

Таблица Г.9 - Наиболее важные элементы класса BinaryReader

Элемент	Описание
BaseStream	Базовый поток, с которым работает объект BinaryReader
Close	Заккрыть поток
PeekChar	Возвратить следующий символ без перемещения внутреннего указателя в потоке
Read	Считать поток байтов или символов и сохранить в массиве, передаваемом как входной параметр
ReadXXXX	Считать из потока данные определенного типа (например, ReadBoolean, ReadByte, ReadInt32 и т. д.)