

# **Лабораторная работа №3**

## **Работа с массивами данных в C# с использованием Windows Forms**

### **1. Цель**

ознакомиться с компонентами и методами работы с массивами в C# с использованием Windows Form в простых Windows приложениях.

### **2. Методические указания**

1. При изучении языка программирования C# будет использоваться интегрированная среда разработки программного обеспечения Microsoft Visual Studio Express 2013 для Windows Desktop. Будут использованы основные элементы .NET Framework и связь C# с элементами платформы .NET.
2. По окончании работы сохраните все созданные файлы на своих носителях.
3. Защита лабораторной работы производится только при наличии электронного варианта работающего скрипта.

### **3. Теоретические сведения**

Алгоритмы и задачи, рассматриваемые в этой лабораторной работе, являются частью фундамента, на котором строится образование программиста. Нет ни одной проблемной области, в задачах которой не требовались бы массивы.

Последовательность элементов –  $a_1, a_2, \dots, a_n$  – одна из любимых структур в математике. Последовательность можно рассматривать как функцию  $a(i)$ , которая по заданному значению индекса элемента возвращает его значение. Эта функция задает отображение  $\text{integer} \rightarrow T$ , где  $T$  – это тип элементов последовательности.

В программировании последовательности называются массивами. Массив – это упорядоченная последовательность элементов одного типа. Порядок элементов задается с помощью индексов.

Для программистов важно то, как массивы хранятся в памяти. Массивы занимают непрерывную область памяти, поэтому, зная адрес начального элемента массива, зная, сколько байтов памяти требуется для хранения одного элемента, и, зная индекс (индексы) некоторого элемента, нетрудно вычислить его адрес, а значит и хранимое по этому адресу значение элемента. На этом основана адресная арифметика в языках C, C++, где адрес элемента  $a(i)$  задается адресным выражением  $a+i$ , в котором имя массива  $a$  воспринимается как адрес первого элемента. При вычислении адреса  $i$ -го элемента индекс  $i$  умножается на длину слова, требуемого для хранения элементов типа  $T$ . Адресная арифметика приводит к 0-базируемости элементов массива, когда индекс первого элемента равен нулю, поскольку первому элементу соответствует адресное выражение  $a+0$ .

Язык C# сохранил 0-базируемость массивов. Индексы элементов массива в языке C# изменяются в плотном интервале значений от нижней границы, всегда равной 0, до верхней границы, заданной динамически вычисляемым выражением, возможно зависящим от переменных. Массивы C# являются 0-базируемыми динамическими массивами. Это важно понимать с самого начала.

Не менее важно понимать и то, что массивы C# относятся к ссылочным типам. Рассмотрим следующий фрагмент программного кода:

```
int[] x, y = {1, 2, 3};  
double[] z;  
int[,] u, v = {{1,3,5},{2,4,6}};
```

Здесь объявлены пять переменных – *x*, *y*, *z*, *u*, *v*. Все они являются массивами, но разных типов. Переменные *x* и *y* принадлежат типу *T* = *int*[], задающему одномерные массивы с элементами целого типа *int*. Переменные *u* и *v* принадлежат другому типу *T1* = *int*[,]- двумерных массивов с элементами целого типа. Переменная *z* принадлежит типу *T3* –одномерных массивов с элементами вещественного типа *double*. Все три типа массивов – *T1*, *T2*, *T3* являются наследниками общего родителя – типа (класса) *Array*, наследуя от родителя многие полезные свойства и методы. Все пять переменных являются типизированными ссылками. В момент объявления три переменные – *x*, *z* и *u* не инициализированы и потому являются «висячими» ссылками со значением *null*. Переменные *y* и *v* объявлены с инициализацией. При инициализации в динамической памяти создаются два объекта соответственно типов *T1* и *T3*, фактически и задающие реальные массивы. У первого из этих объектов 3 элемента, у второго - 6. Ссылки *y* и *v* связываются с этими объектами. При связывании тип ссылки и тип объекта должны быть согласованными. Заметьте, число элементов в массиве является характеристикой объекта, а не типа. Ссылка может быть связана с объектами, содержащими различное число элементов, необходимо лишь выполнение условия согласования типов.

Дополним код следующими строчками:

```
x = new int[10];  
z = new double[20];  
u = new int[3, 5];
```

Здесь последовательно вызываются три конструктора типов *T1*, *T2*, *T3*, создающие новые три объекта в памяти и ссылки *x*, *y*, *z* связываются с этими объектами, так что у массива *x* теперь 10 элементов, *z* – 20, *u* -15.

Рассмотрим, что произойдет в результате присваивания:

```
x = y;  
u = v;
```

Присваивание законно, поскольку переменные в левой и правой части имеют один и тот же (следовательно, согласованный) тип. В результате присваивания переменные порвут связь с теми объектами, с которыми они были связаны, и будут связаны с новыми объектами, так что *x* теперь имеет 3 элемента, *u* – 6.

### **Ввод – вывод массивов**

Как у массивов появляются значения, как они изменяются? Возможны три основных способа:

- вычисление значений в программе;
- значения вводит пользователь;
- связывание с источником данных.

В задачах этой лабораторной работы ограничимся пока рассмотрением первых двух способов. Первый способ более или менее понятен. Простые примеры его применения

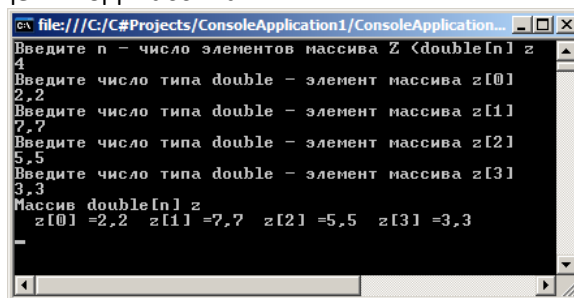
приводились чуть выше. Приведу некоторые рекомендации по вводу и выводу массивов, ориентированные на работу с конечным пользователем.

Для консольных приложений ввод массива обычно проходит несколько этапов:

1. ввод размеров массива;
2. создание массива;
3. организация цикла по числу элементов массива, в теле которого выполняется:
  - приглашение к вводу очередного элемента;
  - ввод элемента;
  - проверка корректности введенного значения.

Вначале у пользователя запрашиваются размеры массива, затем создается массив заданного размера. В цикле по числу элементов организуется ввод значений. Вводу каждого значения предшествует приглашение к вводу с указанием типа вводимого значения, а при необходимости и диапазона, в котором должно находиться требуемое значение. Поскольку ввод значений – это ответственная операция, а на пользователя никогда нельзя положиться, то после ввода часто организуется проверка корректности введенного значения. При некорректном задании значения элемента ввод повторяется, пока не будет достигнут желаемый результат.

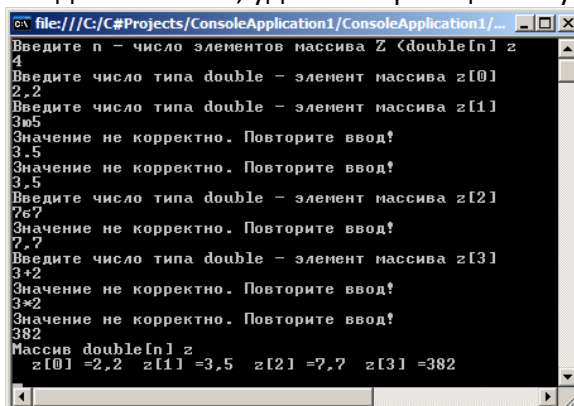
При выводе массива на консоль, обычно вначале выводится имя массива, а затем его элементы в виде пары: <имя> = <значение> (например,  $f[5] = 77,7$ ). Задача усложняется для многомерных массивов, когда для пользователя важно видеть не только значения, но и структуру массива, например располагая одну строку массива в одной строке экрана. Приведу пример того, как может выглядеть ввод – вывод массива в консольном приложении. На рис. 3.1. показан экран в процессе работы консольной программы, организующей ввод массива.



```
file:///C:/C#Projects/ConsoleApplication1/ConsoleApplication1/...
Введите n - число элементов массива Z <double[n]> z
4
Введите число типа double - элемент массива z[0]
2,2
Введите число типа double - элемент массива z[1]
7,7
Введите число типа double - элемент массива z[2]
5,5
Введите число типа double - элемент массива z[3]
3,3
Массив double[n] z
z[0] =2,2 z[1] =7,7 z[2] =5,5 z[3] =3,3
```

Рис. 3.1. Ввод - вывод массива на консоль.

На рис. 3.2. показана работа программы с контролем ввода, где пользователь повторяет ввод, пока не введет значение, удовлетворяющее типу элемента массива.



```
file:///C:/C#Projects/ConsoleApplication1/ConsoleApplication1/...
Введите n - число элементов массива Z <double[n]> z
4
Введите число типа double - элемент массива z[0]
2,2
Введите число типа double - элемент массива z[1]
3*5
Значение не корректно. Повторите ввод!
3,5
Значение не корректно. Повторите ввод!
7,7
Введите число типа double - элемент массива z[2]
7*7
Значение не корректно. Повторите ввод!
7,7
Введите число типа double - элемент массива z[3]
3+2
Значение не корректно. Повторите ввод!
3*2
Значение не корректно. Повторите ввод!
382
Массив double[n] z
z[0] =2,2 z[1] =3,5 z[2] =7,7 z[3] =382
```

Рис. 3.2. Ввод - вывод массива на консоль с контролем значений.

Как организовать контроль ввода? Наиболее разумно использовать для этих целей конструкцию охраняемых блоков – **try – catch** блоков. Это общий подход, когда все опасные действия, обычно связанные с работой пользователя, внешних устройств, внешних источников данных, размещаются в охраняемых блоках. Вот как может выглядеть ввод элемента массива **z[i]** типа **double**, помещенный в охраняемый блок:

```
bool correct;
do
{
    correct = true;
    try
    {
        z[i] = Convert.ToDouble(Console.ReadLine());
    }
    catch (Exception e)
    {
        Console.WriteLine
            ("Значение не корректно. Повторите ввод!");
        correct = false;
    }
} while (!correct);
```

Как правило, для ввода-вывода массивов пишутся специальные процедуры, вызываемые в нужный момент.

### Ввод - вывод массивов в Windows приложениях

Приложения Windows позволяют построить пользовательский интерфейс, облегчающий работу по вводу и выводу массивов. И здесь, когда данные задаются пользователем, заполнение массива проходит через те же этапы, что рассматривались для консольных приложений. Но выглядит все это более наглядно и понятно. Пример подобного интерфейса, обеспечивающего работу по вводу и выводу одномерного массива, показан на рис. 3.3.

Рис. 3.3. Форма для ввода – вывода одномерного массива.

Пользователь вводит в текстовое окно число элементов массива и нажимает командную кнопку «Создать массив», обработчик которой создает массив заданной размерности. Затем он может переходить к следующему этапу – вводу элементов массива. Очередной элемент массива вводится в текстовое окно, а обработчик командной кнопки «Ввести элемент» обеспечивает передачу значения в массив. Корректность ввода можно контролировать и здесь, проверяя значение введенного элемента и выводя в специальное окно сообщение в случае его некорректности, добиваясь, в конечном итоге, получения от пользователя корректного ввода.

Для облегчения работы пользователя текстовое окно для ввода элемента сопровождается специальным окном, содержащим информацию, какой именно элемент должен вводить пользователь. В примере возможного интерфейса пользователя, показанного на рис. 3.3, после того, как все элементы массива введены, окно ввода становится недоступным для ввода элементов. Интерфейс формы позволяет многократно создавать новый массив, повторяя весь процесс.

На рис. 3.3. форма разделена на две части – для ввода и вывода массива. Крайне важно уметь организовать ввод массива, принимая данные от пользователя. Не менее важно уметь отображать существующий массив в форме, удобной для восприятия пользователем. На рисунке показаны три различных элемента управления, пригодные для этих целей – **ListBox**, **CheckedListBox** и **ComboBox**. Программа, форма которой показана на рис. 3.3, сделана так, что как только вводится очередной элемент, он немедленно отображается во всех трех списках.

Отображать массив в трех списках конечно не нужно, это сделано только в целях демонстрации возможностей различных элементов управления. Для целей вывода подходит любой из них, выбор зависит от контекста и предпочтений пользователя. Элемент **CheckedListBox** обладает дополнительными свойствами в сравнении с элементом **ListBox**, позволяя отмечать некоторые элементы списка (массива). Отмеченные пользователем элементы составляют специальную коллекцию. Эта коллекция доступна, с ней можно работать, что иногда весьма полезно.

Ввод двумерного массива немногим отличается от ввода одномерного массива. Сложнее обстоит дело с выводом двумерного массива, если при выводе пытаться отобразить структуру массива. К сожалению все три элемента управления, хорошо справляющиеся с отображением одномерного массива, плохо приспособлены для показа структуры двумерного массива. Хотя у того же элемента **ListBox** есть свойство **MultiColumn**, включение которого позволяет показывать массив в виде строк и столбцов, но это не вполне то, что нужно для наших целей – отображения структуры двумерного массива. Хотелось бы, чтобы элемент имел такие свойства, как **Rows** и **Columns**, а их у элемента **ListBox** нет. Нет их и у элементов **ComboBox** и **CheckedListBox**. Приходится обходиться тем, что есть. На рис. 3.4. показан пример формы, поддерживающей работу по вводу и выводу двумерного массива.

The screenshot shows a Windows application window titled "Form3". It contains two main sections: "Ввод" (Input) and "Вывод" (Output).

In the "Ввод" section, there are two text boxes: "n - строк" (n - rows) with the value "2" and "m - столбцов" (m - columns) with the value "3". To the right of these is a button labeled "Создать массив" (Create array). Below these is a text box for "Ввести элемент" (Enter element) and a button labeled "Ввести элемент". At the bottom of this section are two labels: "Результат ввода" (Input result) and "Ввод недоступен!" (Input unavailable!).

In the "Вывод" section, there is a table displaying the result of the array creation:

1	3	5
2	4	6

Рис. 3.4. Форма, поддерживающая ввод и вывод двумерного массива.

Интерфейс формы схож с тем, что использовался для организации работы с одномерным массивом. Программная настройка размеров элемента управления **ListBox** позволила даже отобразить структуру массива. Но в общей ситуации, когда значения, вводимые пользователем, могут колебаться в широком диапазоне, трудно гарантировать отображение структуры двумерного массива. Однако ситуация не безнадежна. Есть и другие, более мощные и более подходящие для наших целей элементы управления.

### Элемент управления **DataGridView** и отображение массивов

Элемент управления **DataGridView** является последней новинкой в серии табличных элементов **DataGrid**, позволяющих отображать таблицы. Главное назначение этих элементов – связывание с таблицами внешних источников данных, прежде всего с таблицами баз данных. Мы же сейчас рассмотрим другое его применение – в интерфейсе, позволяющем пользователю вводить и отображать матрицы – двумерные массивы.

Рассмотрим классическую задачу умножения прямоугольных матриц  $C=A*B$ .

Построим интерфейс, позволяющий пользователю задавать размеры перемножаемых матриц, вводить данные для исходных матриц **A** и **B**, перемножать матрицы и видеть результаты этой операции. На рис. 3.5. 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.

Рис. 3.5. Форма с элементами DataGridView, поддерживающая работу с матрицами.

На форме расположены три текстовых окна для задания размеров матриц, три элемента **DataGridView** для отображения матриц, три командные кнопки для выполнения операций, доступных пользователю. Кроме того, на форме присутствуют 9 меток (элементов управления **label**), семь из которых видимы на рис. 3.5. В них отображается информация, связанная с формой и отдельными элементами управления. Текст у невидимых на рисунке меток появляется тогда, когда обнаруживается, что пользователь некорректно задал значение какого-либо элемента исходных матриц.

А теперь перейдем к описанию того, как этот интерфейс реализован. В классе **Form2**, которому принадлежит наша форма, зададим поля, определяющие размеры матриц и сами матрицы:

```
//поля класса Form
int m, n, p; //размеры матриц
double[,] A, B, C; //сами матрицы
```

Рассмотрим теперь, как выглядит обработчик события «**Click**» командной кнопки «Создать DataGridView». Предполагается, что пользователь разумен и, прежде чем нажать эту кнопку, задает размеры матриц в соответствующих текстовых окнах. Напомню, что 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 очистку элементов управления **DataGridView**, удаляя предыдущее состояние, затем добавляет столбцы и строки в эти элементы в полном соответствии с заданными размерами матриц. Вот текст обработчика:

```

private void button1_Click(object sender, EventArgs e)
{
    //создание матриц
    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 DGView, 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 DGView столбцами
    AddColumns(n, dataGridView1);
    AddColumns(p, dataGridView2);
    AddColumns(p, dataGridView3);
    //Заполнение DGView строками
    AddRows(m, dataGridView1);
    AddRows(n, dataGridView2);
    AddRows(m, dataGridView3);
}

```

Комментарий. Чистка предыдущего состояния элементов **DataGridView** сводится к удалению столбцов. Продемонстрированы 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 одним махом, - 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**. Вот их текст:

```

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

```

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 нашем примере это имя не используется.

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

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

Рассмотрим теперь, как выглядит обработчик события «Click» следующей командной кнопки «Перенести данные в массив». 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. Текст обработчика:

```

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);
            label8.Text = "";
        }
        catch (Exception any)
        {
            label8.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 < n; i++)
    for (int j = 0; j < p; j++)
    {
        do
        {
            correct = true;
            try
            {
                elem =
dataGridView2.Rows[i].Cells[j].Value.ToString();
                B[i, j] = Convert.ToDouble(elem);
                label9.Text = "";
            }
            catch (Exception any)
            {
                label9.Text = "Значение элемента" +
                    "B[" + i.ToString() + ", " + j.ToString() + "]"
                    + " не корректно. Повторите ввод!";
                dataGridView2.Rows[i].Cells[j].Selected=true;

                Form3 frm = new Form3();
                frm.label1.Text =
                    "B[" + i.ToString() + ", " + j.ToString() + "]= ";
                frm.ShowDialog();
                dataGridView2.Rows[i].Cells[j].Value = frm.textBox1.Text;
                correct = false;
            }
        } while (!correct);
    }
}

```

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.

Как всегда при вводе основной проблемой является обеспечение корректности вводимых данных. Схема, рассматриваемая нами ранее, нуждается в корректировке. Дело в том, что ранее проверка корректности осуществлялась сразу же после ввода пользователем значения элемента. Теперь 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 предлагаются два различных подхода к решению этой проблемы.

Первый подход демонстрируется на примере ввода элементов матрицы **A**. Как обычно, 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)**. Заметьте, 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. (Помните, 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». Этот подход понятен и легко реализуем. Недостатком является его неэффективность, поскольку повторно будут переноситься в массив все элементы, в том числе и те, что были введены вполне корректно. У программиста такая ситуация может вызывать чувство неудовлетворенности своей работой.

На примере ввода элементов матрицы **B** продемонстрируем другой подход, когда исправляется только некорректно заданное значение. Прежде, чем читать дальше, попробуйте найти собственное решение этой задачи. Это оказывается не так просто, как может показаться с первого взгляда. Для организации диалога с пользователем пришлось организовать специальное диалоговое окно, представляющее обычную форму с двумя элементами управления – меткой для выдачи информационного сообщения и текстовым окном для ввода пользователем корректного значения. При обнаружении ошибки ввода открывается диалоговое окно, в которое пользователь вводит корректное значение элемента и закрывает окно диалога. Введенное пользователем значение переносится в нужную ячейку таблицы **DataGridView**, а оттуда в матрицу.

При проектировании диалогового окна значение свойства формы **FormBorderStyle**, установленное по умолчанию как «sizeable» следует заменить значением «FixedDialog», что влияет на внешний вид и поведение формы. Важно отметить, что форма, представляющее диалоговое окно, должна вызываться не методом **Show**, а методом **ShowDialog**. Иначе произойдет заикливание, начнут порождаться десятки диалоговых окон, прежде чем успеете нажать спасительную в таких случаях комбинацию **Ctrl+ Alt + Del**.

Приведем снимки экранов, демонстрирующие ситуации, в которых пользователь ввел некорректные значения. На рис. 3.6. показано информационное сообщение, появляющееся при обнаружении некорректного ввода значений элементов матрицы **A**.

Умножение матриц

Задайте размеры матриц!

m =  n =  p =

Матрица A (m\*n)

	Column0	Column1	Column2
row0	2	2.5	3
row1	4	4*5	4
*			

Значение элементаA[0, 1] не корректно. Повторите ввод!

Матрица B (n\*p)

	Column0	Column1
row0	2	3
row1	4*5	2+2
row2	3	4
*		

Матрица C (m\*p)

	Column0	Column1
row0		
row1		
*		

Создать DataGridView

Перенести данные в массив

Умножить матрицы

Рис. 3.6. Информационное сообщение о некорректных значениях матрицы A.

После появления подобного сообщения пользователь должен исправить некорректное значение (одно или несколько) и повторить процесс переноса данных. На рис. 3.7. показана ситуация, когда некорректно заданное значение исправляется в открывшемся окне диалога.

Корректировка элемента

B[1, 0] =

матриц!

m =  n =  p =

Матрица A (m\*n)

	Column0	Column1	Column2
row0	2	2.5	3
row1	4	4.5	4
*			

Матрица B (n\*p)

	Column0	Column1
row0	2	3
row1	4*5	2+2
row2	3	4
*		

Значение элементаB[1, 0] не корректно. Повторите ввод!

Матрица C (m\*p)

	Column0	Column1
row0		
row1		
*		

Создать DataGridView

Перенести данные в массив

Умножить матрицы

Рис. 3.7. Диалоговое окно для корректировки значений элементов матрицы B.

Обработчик события **«Click»** командной кнопки «Умножить матрицы» выполняет ответственные задачи – реализует умножение матриц и отображает полученный результат в таблице соответствующего элемента **DataGridView**. Но оба эти действия выполняются естественным образом, не требуя кроме циклов никаких специальных средств и программистских ухищрений. Программный код без дополнительных комментариев:

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

#### 4. Содержание отчёта

Отчёт должен содержать название и цель. Ход работы с комментариями и выполненное задания из ПРИЛОЖЕНИЕ А. Выводы.

#### 5. Контрольные вопросы

1. Что такое массивы?
2. Особенности работы с массивами в C#.
3. Компоненты работы с массивами в Windows Form
4. Методы заполнения двумерных массивов в Windows Form
5. Опишите принципы работы ListBox в Windows Form
6. Опишите принципы работы CheckedListBox в Windows Form
7. Опишите принципы работы ComboBox в Windows Form
8. Опишите принципы работы DataGridView в Windows Form

1. Дан массив размера  $N$  и целые числа  $K$  и  $L$  ( $1 < K \leq L \leq N$ ). Найти сумму всех элементов массива, кроме элементов с номерами от  $K$  до  $L$  включительно. Организуйте в Windows приложении ввод массива и вывод результата.
2. Организуйте в Windows приложении ввод массива «Сотрудники», содержащего фамилии сотрудников. Введите массив «Заявка», элементы которого содержат фамилии сотрудников и, следовательно, должны содержаться в массиве сотрудников. Обеспечьте контроль корректности ввода данных и выведите массив «Заявка».
3. Организуйте в Windows приложении ввод массива «Сотрудники», содержащего фамилии сотрудников. Создайте массив «Заявка», элементы которого должны содержаться в массиве сотрудников. Для создания массива «Заявка» постройте форму «Два списка», содержащую два элемента ListBox, источником данных для первого из них служит массив «Сотрудники». Пользователь переносит данные из первого списка во второй, формируя данные для массива «Заявка». После формирования данные переносятся в массив.
4. Организуйте в Windows приложении ввод массива «Студенты» из двух столбцов, содержащего фамилии и имена студентов. Введите массив «Общежитие» той же структуры, элементы которого должны содержаться в массиве сотрудников. Обеспечьте контроль корректности ввода данных. Организуйте вывод обоих массивов.
5. Организуйте в Windows приложении ввод и вывод массива «Машины», содержащего 4 столбца: «Владелец», «Марка», «Номер», «Год Выпуска». При вводе данных обеспечьте их корректность. Поле «Владелец» должно быть строкой в формате «фамилия имя», где фамилия и имя должны начинаться с большой буквы и состоять из букв алфавита кириллицы, включая дефис. Номер машины должен соответствовать формату, принятому для номеров машин. При выводе сохраняйте структуру массива.
6. Организуйте в Windows приложении ввод массива «Студенты», содержащего фамилии студентов, и массива «Стипендия». Обеспечьте контроль корректности ввода данных о стипендии, проверяя диапазон возможных значений, диапазон задается в полях на форме.
7. Организуйте в Windows приложении ввод и вывод матрицы - двумерного массива арифметического типа. Реализовать систему автоматического заполнения массива случайными числами.
8. Организуйте в Windows приложении ввод массива декартовых координат  $n$  точек на плоскости. Вычислите массив полярных

координат этих точек и организуйте вывод этого массива. Обеспечьте контроль вводимых значений.

9. Дана матрица размера  $M \times N$ . Поменять местами столбец с номером  $N$  и последний из столбцов, содержащих только положительные элементы. Если требуемых столбцов нет, то вывести матрицу без изменений.
10. Организуйте в Windows приложении ввод массива полярных координат  $n$  точек на плоскости. Вычислите массив декартовых координат этих точек и организуйте вывод этого массива. Обеспечьте контроль вводимых значений.
11. Организуйте в Windows приложении ввод массива декартовых координат  $n$  точек в трехмерном пространстве. Вычислите массив полярных координат этих точек и организуйте вывод этого массива. Обеспечьте контроль вводимых значений.
12. Даны целые положительные числа  $M$  и  $N$ . Сформировать целочисленную матрицу размера  $M \times N$ , у которой все элементы  $J$ -го столбца имеют значение  $5 \cdot J$  ( $J = 1, \dots, N$ ). Обеспечьте контроль вводимых значений. Заполнить массив случайными числами и вывести его в соответствии с заданием.
13. Заполнить матрицу случайными числами размера  $M \times N$  и целое число  $K$  ( $1 \leq K \leq M$ ). Вывести элементы  $K$ -й строки данной матрицы. Обеспечьте контроль вводимых значений. Обеспечьте контроль вводимых значений.
14. Дана матрица размера  $M \times N$ . Для каждой строки матрицы найти сумму ее элементов. Обеспечьте контроль вводимых значений.
15. Дана матрица размера  $M \times N$ . В каждом столбце матрицы найти максимальный элемент. Обеспечьте контроль вводимых значений.
16. Дана матрица размера  $M \times N$ . В каждой ее строке найти количество элементов, меньших среднего арифметического всех элементов этой строки. Обеспечьте контроль вводимых значений.
17. Дана матрица размера  $M \times N$  и целые числа  $K_1$  и  $K_2$  ( $1 \leq K_1 < K_2 \leq M$ ). Поменять местами строки матрицы с номерами  $K_1$  и  $K_2$ . Обеспечьте контроль вводимых значений.
18. Дана матрица размера  $M \times N$ . Преобразовать матрицу, поменяв местами минимальный и максимальный элемент в каждой строке.
19. Дана матрица размера  $M \times N$  и целое число  $K$  ( $1 \leq K \leq N$ ). Удалить столбец матрицы с номером  $K$ .
20. Дана матрица размера  $M \times N$  и целое число  $K$  ( $1 \leq K \leq M$ ). Перед строкой матрицы с номером  $K$  вставить строку из значений  $S$ . Обеспечьте контроль вводимых значений.