ВВЕДЕНИЕ

Visual Studio 2013 - созданная компанией Microsoft интегрированная среда разработки (IDE) для создания, запуска и отладки приложений на различных языках программирования .NET.

Создание нового проекта

Разработка приложения в Visual C# начинается с создания нового или открытия существующего проекта. Чтобы создать новый проект, выполните команду Файл->Создать проект , а чтобы открыть существующий - команду Файл->Открыть проект... Проект представляет собой группу взаимосвязанных файлов. В Visual C# приложения организуются в проекты (projects) и решения (solutions), содержащие один или несколько проектов. Решения используются при создании крупномасштабных приложений.

Диалоговое окно New Project и шаблоны проектов

При выполнении команды Файл->Создать проект на начальной странице открывается диалоговое окно Создать проект (рис. 1.1). Visual Studio предоставляет разработчику несколько шаблонов (см. рис. 1.1) - типов проектов, которые могут создаваться в Visual C# и других языках. В их число входят шаблоны приложений Windows Forms, приложений WPF и многих других - полная версия Visual Studio содержит много дополнительных шаблонов. Приложение Windows Forms выполняется в операционной системе Windows (например,Windows 7 или Windows 8) и обычно обладает графическим интерфейсом (GUI), с которым взаимодействуют пользователи. По умолчанию Visual Studio присваивает новому проекту и решению на базе шаблона Windows Forms Application имя WindowsFormsApplication1 (см. рис. 1.1).

Выберите шаблон Windows Forms и щелкните на кнопке ОК, чтобы перевести IDE в режим конструирования (рис 1.2). Функции этого режима предназначены для создания графического интерфейса приложения. Не забудьте переименовать проект и указать его расположение).

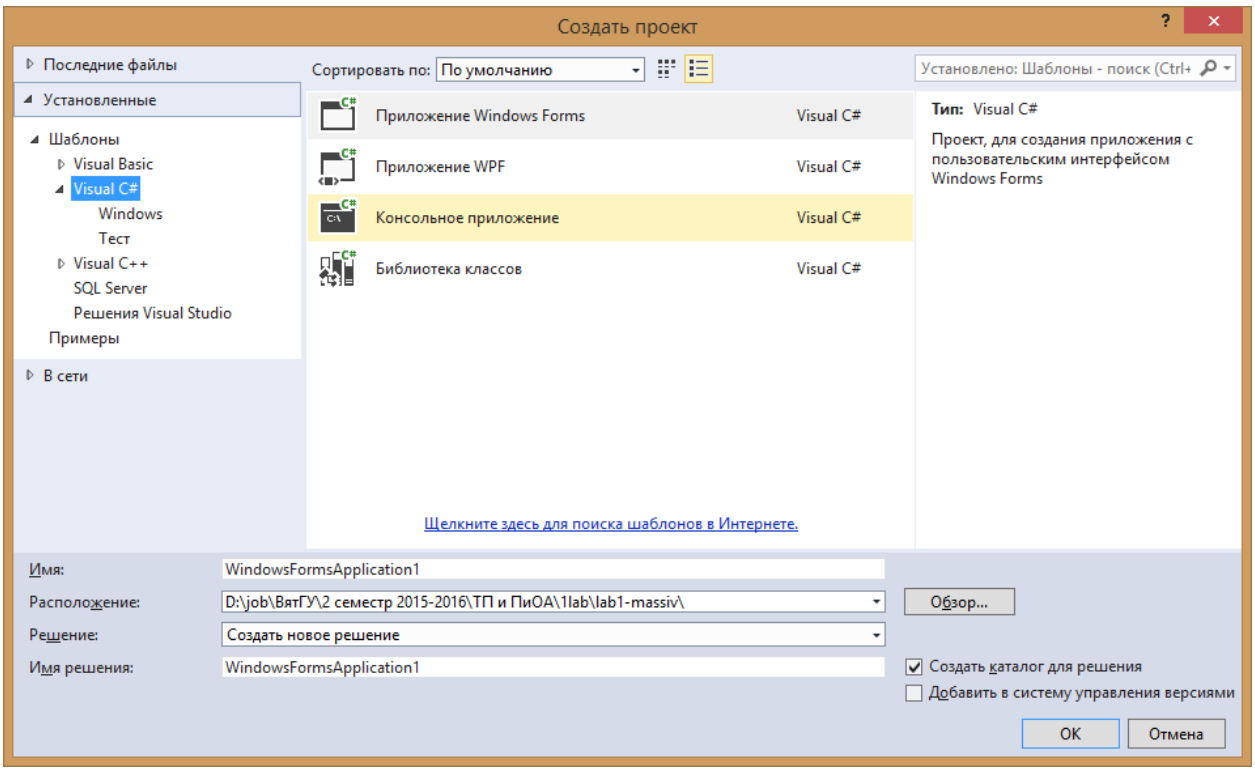

Рисунок 1.1 – Диалоговое окно Создать проект

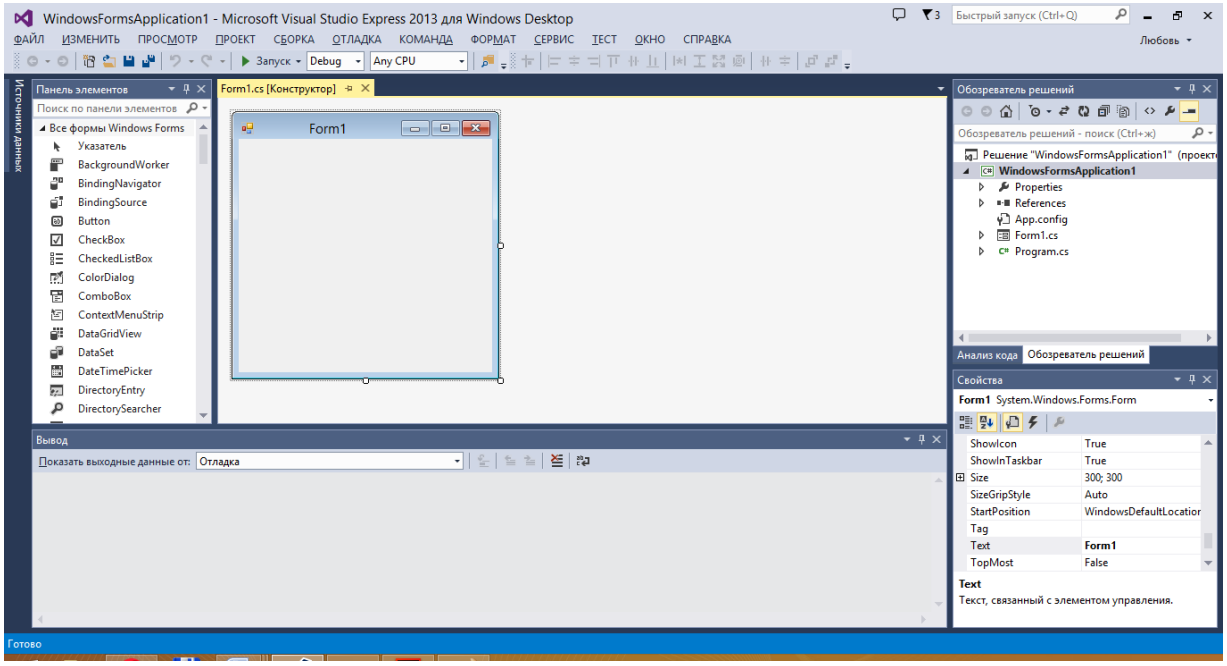

Рисунок 1.2 – Режим конструирования (Design view) в IDE

Формы и элементы управления

Прямоугольник с заголовком Forml (форма) представляет главное окно создаваемого приложения Windows Forms. Приложения Visual C# могут содержать несколько форм (окон). Формы можно изменять, добавляя к ним элементы управления, например надпись (Label) и графическое поле (PictureBox). Надписи обычно содержат текст с описанием, а в графическом поле выводится изображение. Visual Studio содержит много готовых

элементов управления и других компонентов, которые могут использоваться для создания и настройки поведения приложений.

Разместив элемент на форме, разработчик может изменить его Свойства либо программно, либо вручную, выбрав соответствующую позицию в панели свойств и задав требуемые настройки. Форма и элементы управления образуют графический интерфейс приложения. Пользователь набирает данные на клавиатуре, щелкает кнопками мыши или вводит их другими способами. Графический интерфейс используется для отображения инструкций и другой информации, предназначенной для пользователя. Имя каждого открытого документа выводится на корешке вкладки. Чтобы просмотреть документ при наличии нескольких открытых документов, щелкните на соответствующем корешке. Активная вкладка (вкладка с документом, отображаемым в настоящее время) выделена синим цветом (например, Forml.cs [Конструктор] на рисунке 1.2.).

Перемещение в Visual Studio IDE

IDE предоставляет окна для работы с файлами проекта и настройки элементов управления. В этом разделе описаны некоторые окна, часто используемые при разработке приложений Visual C#. Чтобы обратиться к любому из этих окон, выберите его имя в меню Просмотр.

Solution Explorer

Окно Solution Explorer (рисунок 1.3) предоставляет доступ ко всем файлам приложения. Если оно не отображается в IDE, выполните команду Просмотр->Обозреватель решений. Когда вы открываете новое или существующее решение, его содержимое выводится в окне Solution Explorer.

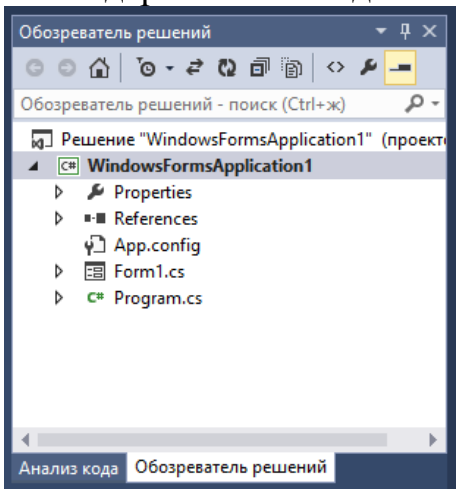

Рисунок 1.3 – Окно Solution Explorer с открытым проектом

Для решений с одним проектом, стартовый проект является единственным (в данном случае WindowsFormsApplication1). Имя стартового проекта выделяется жирным шрифтом в окне Solution Explorer. При первом создании приложения окно Solution Explorer выглядит так, как показано на рисунке 1.3. Файл Visual C#, соответствующий форме называется Forml.cs. Файлы Visual C# используют расширение .cs. По умолчанию в IDE

отображаются только те файлы, которые вам, возможно, придется редактировать; другие файлы, сгенерированные IDE, остаются скрытыми. В окне Solution Explorer присутствует панель инструментов с несколькими кнопками. Кнопка Show AII Files отображает все файлы решения, включая сгенерированные IDE. Щелчок на стрелке слева от узла разворачивает или сворачивает этот узел. Эта схема также используется в других окнах Visual Studio.

Панель элементов

Чтобы вызвать окно панели элементов (Toolbox), выполните команду Просмотр->Панель элементов. В нем содержатся элементы, используемые для модификации форм (рис. 1.4). В методологии визуального программирования разработчик <<перетаскивает» элементы управления на форму, а IDE генерирует код создания этих элементов. Так код создается проще и быстрее, чем при самостоятельном написании.

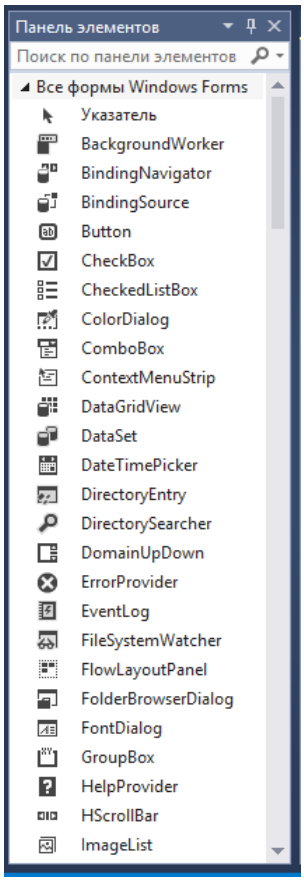

Рисунок 1.4 – Окно «Панель элементов» с элементами управления

Готовые элементы управления на панели элементов сгруппированы по категориям

Окно свойств

Если окно свойств (Properties) не отображается под окном Solution Explorer, выполните команду Просмотр->Окно свойств.. В окне свойств (рис. 1.5) выводятся свойства текущей выделенной формы, элемента управления или файла. Свойства описывают характеристики формы или

элемента управления: размер, цвет, позицию и т. д. Разные формы и элементы управления обладают разными наборами свойств – описание свойства выводится в нижней части окна свойств при выделении этого свойства.

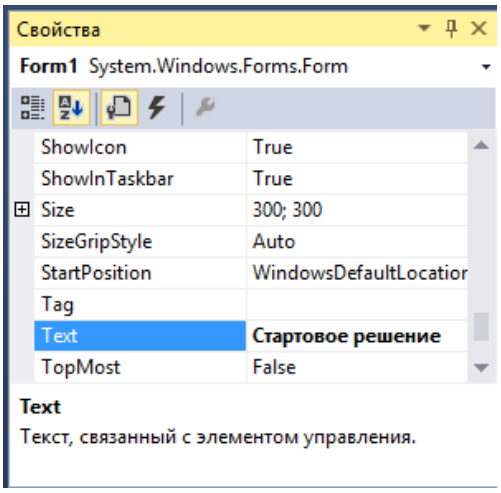

Рисунок 1.5 – Окно свойств

На рисунке 1.5 изображено окно свойств формы Forml. В левом столбце перечислены свойства формы, а в правом выводятся текущие значения каждого свойства. Список свойств можно отсортировать либо по алфавиту (кнопка Alphabetical), либо по категориям (кнопка Categorized). В зависимости от размера окна свойств некоторые свойства могут не поместиться на экране. Чтобы прокрутить список свойств, используйте полосу прокрутки или кнопки со стрелками у верхнего и нижнего края полосы. В окне свойств выводится краткое описание выделенного свойства, помогающее понять его предназначение. Свойство можно быстро задать в этом окне, без написания какого-либо кода. В верхней части окна свойств расположен раскрывающийся список для выбора компонентов. В нем можно выбрать форму или элемент управления, свойства которых вы хотите вывести в окне свойств, без выделения этой формы/элемента управления в графическом интерфейсе.

ЛАБОРАТОРНАЯ РАБОТА 1

ФОРМИРОВАНИЕ И ОБРАБОТКА ЧИСЛОВЫХ МАССИВОВ

ОДНОМЕРНЫЕ МАССИВЫ

Массивы занимают место в памяти. Так как они являются объектами, обычно они создаются ключевым словом new. При создании объекта массива тип и количество его элементов указываются в выражении создания массива, использующем ключевое слово new. Такое выражение возвращает ссылку, которая сохраняется в переменной массива.

Следующее объявление и выражение создания массива создает объект массива с 12 элементами и сохраняет ссылку на него в переменной c:

```
int[] c= new int[ 12 ];
```

Инициализация элементов массива.

```
int[] c; // Объявление переменной массива
c= new int[ 12 ]; // Создание массива; присваивание переменной массива
```

В этом объявлении квадратные скобки за типом int указывают, что переменная c будет ссылаться на массив с элементами int (то есть в c будет храниться ссылка на объект массива). Во второй команде переменной массива c присваивается ссылка на новый объект массива с 12 элементами int. Количество элементов также может задаваться выражением, вычисляемым на стадии выполнения. При создании массива каждому элементу присваивается значение по умолчанию - 0 для простых числовых типов, false для элементов bool и null для ссылок.

Массивы имеют фиксированную длину, вы можете использовать статический метод Resize класса Array с двумя аргументами (массив и новая длина) для создания нового массива заданной длины. Метод копирует содержимое старого массива в новый массив и задает переменной, полученной в первом аргументе, ссылку на новый массив.

```
int[] newArray = new int[ 5 ];
Array.Resize( ref newArray, 10 );
```

Переменная newArray изначально содержит ссылку на массив из 5 элементов. В результате вызова метода Resize переменная newArray начинает указывать на новый массив из 10 элементов. Если новый массив меньше старого, то все данные, не поместившиеся в новый массив, усекаются без каких-либо предупреждений.

Передача массивов и элементов массивов методам

Чтобы передать методу **аргумент-массив**, укажите имя массива без квадратных скобок. Допустим, массив abracadabra объявляется в следующем виде:

```
double[ ] abracadabra = new double [ 24 ];
```

Вызов метода

```
ModifyAbracadabra (abracadabra);
```

передает ссылку на массив abracadabra методу ModifyAbracadabra. Каждый объект

массива «знает» свою длину (и предоставляет доступ к ней в свойстве Length). Таким

образом, при передаче методу ссылки.

Чтобы метод получал **ссылку на массив** при вызове метода, в списке параметров метода должен быть указан параметр-массив. Например, заголовок метода ModifyAbracadabra должен быть записан в виде

```
void ModifyAbracadabra (double [ ] b)
```

Он показывает, что ModifyAbracadabra получает в параметре b ссылку на массив с элементами double. Вызов метода передает ссылку на массив abracadabra, так что при использовании переменной b в вызванном методе она ссылается на тот же объект массива, что и переменная abracadabra в вызывающем методе.

ДВУМЕРНЫЕ МАССИВЫ (МАТРИЦЫ)

Двумерные массивы часто используются для представления таблиц с информацией, упорядоченной по строкам и столбцам. Конкретный элемент таблицы идентифицируется двумя индексами. Обычно первый индекс определяет строку, а второй - столбец, на пересечении которых находится элемент. Массивы, у которых элемент определяется двумя индексами, называются двумерными массивами. В C# поддерживаются два типа двумерных массивов – прямоугольные и ступенчатые.

Прямоугольные массивы

Прямоугольные массивы используются для представления таблиц, организованных в форме строк и столбцов, у которых каждая строка содержит одинаковое число столбцов. Обычно двумерный массив с т строк и п столбцов называется массивом т х п.

Каждый элемент массива идентифицируется выражением а [строка, столбец];, где а - имя массива, а строка и столбец - индексы, однозначно идентифицирующие каждый элемент массива а номерами строки и столбца.

Инициализатор двумерного прямоугольного массива

Многомерные массивы (как и одномерные) могут инициализироваться при объявлении.

Прямоугольный массив b из двух строк и двух столбцов объявляется и инициализируется вложенными инициализаторами массивов:

```
int [ , ] b = { { 1, 2 } , { 3, 4 } } ;
```

Прямоугольный массив может создаваться традиционным выражением создания массива. Например, следующий фрагмент объявляет переменную Ь и присваивает ей ссылку на прямоугольный массив 3 х 4:

```
int[ , ] b;
b = new int[ 3, 4 ];
```

В этом случае количества строк и столбцов задаются литералами 3 и 4, но это не обязательно - размеры массивов могут также задаваться переменными и выражениями.

Как и в случае с одномерными массивами, элементы прямоугольного

массива инициализируются при создании объекта массива.

Ступенчатые массивы

Ступенчатый массив хранится как одномерный массив, в котором каждый элемент ссылается на одномерный массив. Этот способ представления ступенчатых массивов делает их чрезвычайно гибкими, потому что длины строк массива могут быть разными.

Задания

1. В одномерном массиве нулевые элементы удалить, положительные элементы расставить по убыванию, отрицательные - по возрастанию. Получить зависимость затрат машинного времени от размера массива.
2. В матрице удалить строки с последними отрицательными элементами, а затем добавить строку из сумм элементов по столбцам.

Проектирование приложения. Выбор, размещение и задание свойств компонентов. Коды обработчиков событий и функций

1. Запустите VS.
2. Создайте новый проект.
3. Выделите форму, щелкнув на ней левой кнопкой мыши, и в свойство **Text** впишите *МАССИВЫ*.
4. Вначале перенесите на форму многостраничную панель – компонент **tabControl1**. Для размещения остальных компонентов приложения достаточно использовать две страницы компонента – по заданиям 1 и 2 соответственно.
5. Щелкните на компоненте **tabControl1** правой кнопкой мыши и во всплывшем меню используйте команду *Добавить вкладку*. В свойство **Text** первой страницы впишите *массив,* второй – *матрица*..
6. Перенесите на первую страницу (*массив*) компонент **tabControl2** и, как и в **tabControl1**, создайте две страницы, с надписями *тестирование и использование* и *графики* соответственно.

7. На страницу *тестирование и использование* (рис.2.1) перенесите семь меток **Label** (страница *Стандарт*), в свойство **Text** (надпись) которых соответственно впишите *размер массива, начальный, конечный, шаг, диапазон чисел, макс, мин*; пять компонентов ввода целых чисел – **numericUpDown1** для задания параметров формируемых массивов. В свойствах **Minimum** и **Maximum** укажите необходимый вам диапазон значений.

8. Для разрешения или запрещения вывода на экран исходных и результирующих массивов, а также графиков полученных зависимостей, в правую верхнюю часть страницы перенесите два компонента-индикатора **checkBox**, в свойство **Text** которых впишите соответственно *в таблицу* и *графики*.

9. Ниже индикаторов **checkBox** поместите панель **Label** для вывода в нее сообщений: *Макс значение не м.б. меньше мин значения!, В массиве*

только нули!, Кол-во сравнений = <число> Кол-во обменов = <число> . Очистите свойство **Text** у панели.

10. Для вывода массивов на экран при тестировании перенесите на страницу компонент **dataGridView1.**

11. Разместите 2 кнопки с надписями соответственно: **Button1 –** *ПУСК*, **Button2 –** *СБРОС*.

12. Для отображения процесса обработки массива ниже компонента **dataGridView1** поместите компонент **progressBar1**.

13. Далее поместите компонент **statusStrip1.** Щелкните левой клавишей мышки и в выпадающем меню на компоненте и выберите **StatusLabel.** В свойстве **Text** элемента **toolStripStatusLabel1** введите *«Нули удалить, положительные элементы расставить по убыванию, а отрицательные по возрастанию»*.

14. Затраты машинного времени на обработку массива оценивают косвенно – по количествам сравнений и обменов. Для вывода на экран зависимостей количеств сравнений и обменов от размера массива, на страницу *графики* (рис.2.2) компонента **tabControl2** перенесите компоненты **chart1** и **chart2.**

15. Задайте свойства компонентов самостоятельно.

16. После проектирования формы будут иметь следующий вид (рис. 2.1-2.2):

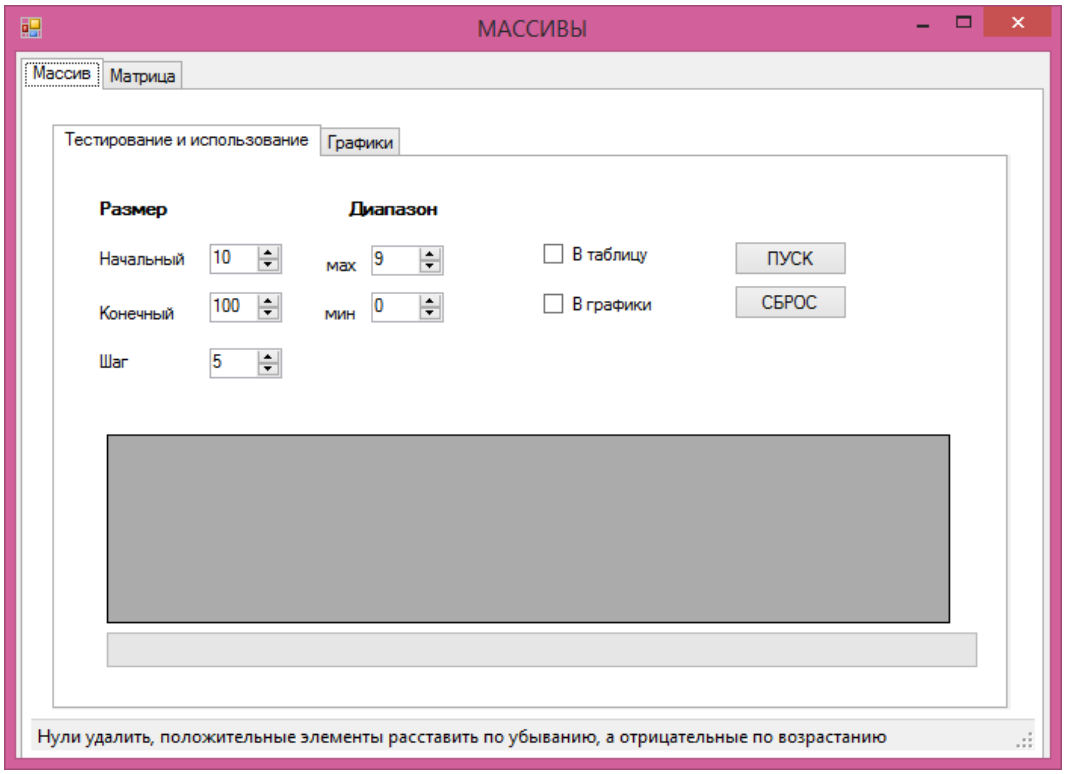

Рисунок 2.1 – форма по окончании проектирования (вид 1)

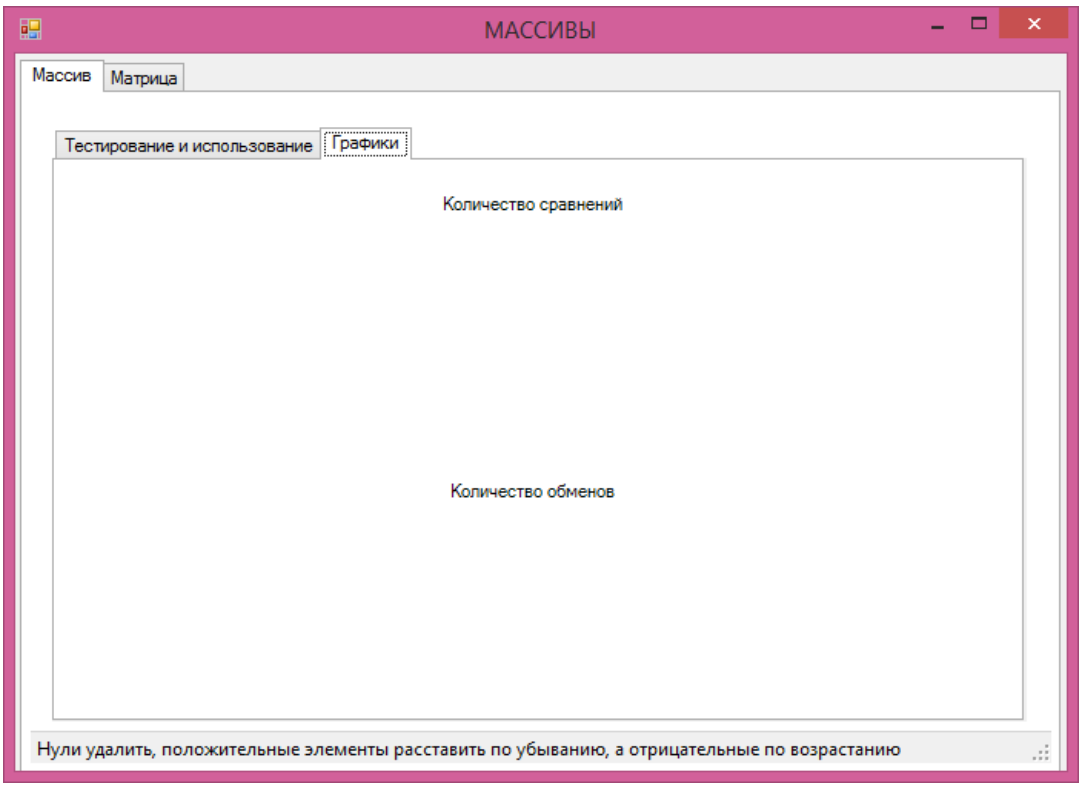

Рисунок 2.2 – форма по окончании проектирования (вид 2)

17. Перейдем к обработчикам событий – щелчков на кнопках ПУСК1 и СБРОС. В первом обработчике размещен алгоритм формирования и обработки массива по заданию 1, а также вывод сообщений и результатов в виде таблицы и графиков, а во втором – подготовка компонентов к выводу новых сообщений и новых результатов. Перед, после и в обработчике щелчка на кнопке ПУСК напишите:

```
int i = 0;
        private void button1_Click(object sender, EventArgs e)
        {
           button2.Enabled = false;
           label8.Text = "";
           if (numericUpDown4.Value < numericUpDown5.Value)
           { label8.Text = "Макс значение не м.б. меньше мин
значения!"; return; }
           int count, current = 0;
           count = (Convert.ToInt32(numericUpDown2.Value) -
Convert.ToInt32(numericUpDown1.Value)) /
Convert.ToInt32(numericUpDown3.Value) + 1;
           for (int n = Convert.ToInt32(numericUpDown1.Value); n <=
Convert.ToInt32(numericUpDown2.Value); n +=
Convert.ToInt32(numericUpDown3.Value))
           {
               int[] vptr=new int[n];
               Random rand = new Random();
               for(int j=0;j<n;j++)
                 {
```

```
                    vptr[j] =
rand.Next(Convert.ToInt32(numericUpDown5.Value),
Convert.ToInt32(numericUpDown4.Value));
                  }
                if(checkBox1.Checked)
                 {
                   dataGridView1.ColumnCount = n+1;
                   dataGridView1.Rows.Add();
                   dataGridView1.Rows[i].Cells[0].Value = "Исходный
массив";
                   for (int j = 0; j < n; j++)
                          { dataGridView1.Rows[i].Cells[j + 1].Value =
vptr[j]; }
                   i++;
                 }
                sort(vptr,n);
                current += 1;
                  progressBar1.Value = 100 * current / count;
            }
      }
        private void sort(int[] p, int n)
        {
         int k = 0, sr = 0, obm = 0, m = 0;
            for (int j = 0; j < n; j++)
            {
                if (p[j] == 0) k++;
                else p[j - k] = p[j];
            }
            n -= k;
            sr += n;
            if (n == 0) { label8.Text = "В массиве одни нули"; return;
}
            for (m = 0; m < n - 1; m++)
                for (int j = m + 1; j < n; j++)
                {
                    if (p[m] > 0 && p[j] > 0 && p[m] < p[j]) {
swap(ref p[m], ref p[j]); obm++; }
                    if (p[m] < 0 && p[j] < 0 && p[m] > p[j]) {
swap(ref p[m], ref p[j]); obm++; }
                    sr += 6;
                }
            if (checkBox1.Checked)
            {
                dataGridView1.AutoResizeColumns();
                dataGridView1.Rows.Add();
                dataGridView1.Rows[i].Cells[0].Value = "Получен
массив";
                for (int j = 0; j < n; j++)
                { dataGridView1.Rows[i].Cells[j + 1].Value = p[j]; }
                i++;
            }
```

```
            if (Convert.ToInt32(numericUpDown1.Value) ==
Convert.ToInt32(numericUpDown2.Value))
            { label8.Text = "Количество сравнений=" +
Convert.ToString(sr) + " Количество обменов=" + Convert.ToString(obm);
}
            if (checkBox2.Checked)
            {
                chart1.Series[0].Points.AddXY(n, sr);
                chart2.Series[0].Points.AddXY(n, obm);
            }
         }


        void swap(ref int x, ref int y)
        { int z = x; x = y; y = z; }
```

18. В обработчике щелчка на кнопке СБРОС напишите:

```
private void button1_Click(object sender, EventArgs e)
        {
            chart1.Series[0].Points.Clear();
            chart2.Series[0].Points.Clear();
            dataGridView1.Rows.Clear();
            dataGridView1.Columns.Clear();
            i = 0;
            button2.Enabled = true;
            }
```

19. Перенесите на вторую страницу (*матрица*) компонента **tabControl1** (рис.2.3) восемь меток **Label**,в свойство **Text** (надпись) которых впишите значения *размерность, диапазон чисел, макс, мин, исходная матрица, матрица-результат*; три компонента ввода целых чисел – **numericUpDown** для задания параметров формируемых матриц.

20. Перенесите также две кнопки **Button** с надписями *ПУСК, СБРОС*.

21. Сюда же поместите панель **Label** для вывода в панель сообщений: *Макс не м.б. меньше мин!, В матрице удалены все строки!, В матрице нет удаленных строк!, В матрице удалено <кол-во> строк(и)!*. Очистите свойство **Text** у панели.

22. Кроме того, на этой странице разместите две таблицы - компоненты **dataGridView** - для вывода матриц – исходной и матрицы-результата.

23. Далее поместите компонент **statusStrip2.** Щелкните левой клавишей мышки и в выпадающем меню на компоненте и выберите **StatusLabel.** В свойстве **Text** элемента **toolStripStatusLabel2** введите *«Строки с "-" на конце - удалить, а затем добавить строку из сумм по столбцам»*.

24. После проектирования формы будут иметь следующий вид (рис. 2.3):

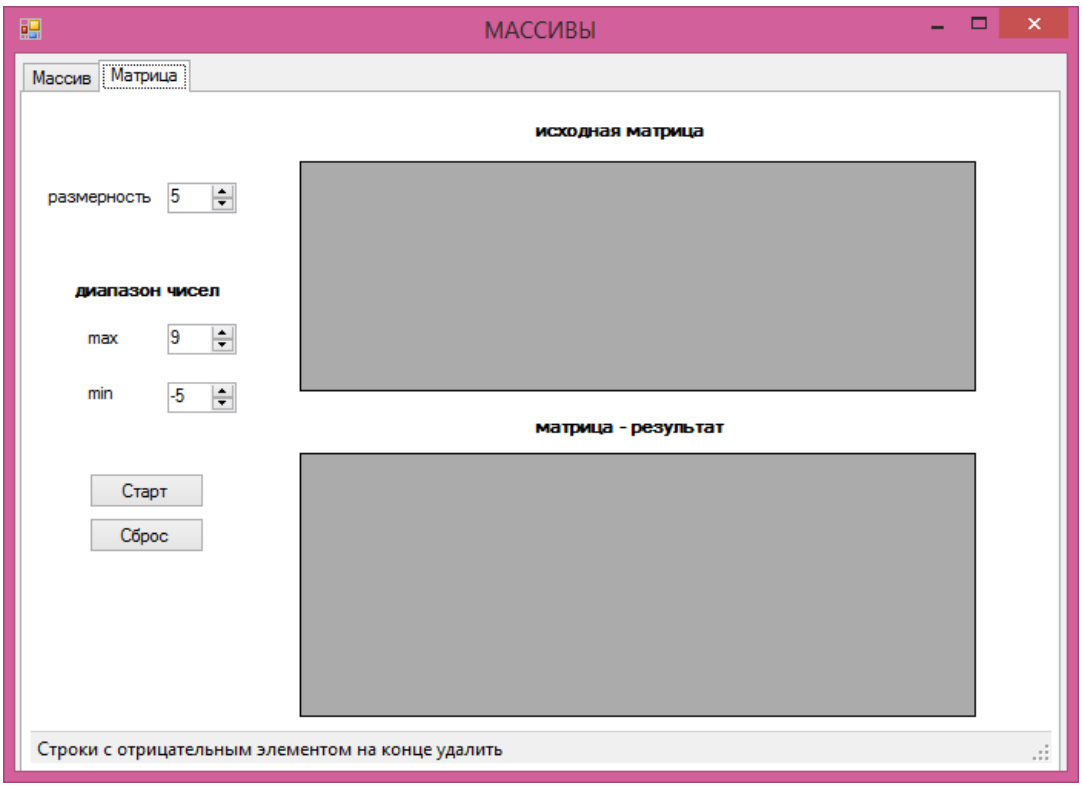

Рисунок 2.3 – форма по окончании проектирования (вид 2)

25. В обработчике щелчка на кнопке *ПУСК* находится алгоритм формирования и обработки динамической матрицы согласно заданию 2 с выводом результатов и сообщений, а в обработчике щелчка на кнопке *СБРОС* - зачистка выведенных результатов и сообщений:

```
int n, m;
        private void button3_Click(object sender, EventArgs e)
        {
            int i, j, k, q;
            button3.Enabled = false;
            if (numericUpDown8.Value < numericUpDown9.Value)
            { label9.Text = "Макс значение не м.б. меньше мин
значения!"; return; }
            n = Convert.ToInt32(numericUpDown6.Value);
            m = Convert.ToInt32(numericUpDown6.Value);
            int[,] ptr;
            ptr = new int[m, n];
            Random rand = new Random();
            dataGridView2.AutoResizeColumns();
            dataGridView2.ColumnCount = n;
            for (i = 0; i < n; i++)
            {
                dataGridView2.Rows.Add();
                for (j = 0; j < m; j++)
                {
                    ptr[i,j] =
rand.Next(Convert.ToInt32(numericUpDown9.Value),
Convert.ToInt32(numericUpDown8.Value));
```

```
                    dataGridView2.Rows[i].Cells[j].Value = ptr[i,j];
                }
            }
            q = 0;
            k = 0;
           if (ptr[n - 1, m - 1] < 0) k++;
         for(q=0;q<n-1;q++)

            {
                if (ptr[q, m - 1] < 0)
                {
                    k++;
                    for (i = q; i < n - 1; i++)
                    {
                    for (j = 0; j < m; j++) ptr[i, j] = ptr[i + 1, j];

                    }
                    q--;
                }
                if (k + q+1 == n) { break; }
            }

         if (k == n) { label9.Text = "В матрице удалены все строки";
return; }
         if (k == 0) { label9.Text = "В матрице нет удаленных строк";
return; }
         label9.Text = "В матрице удалено " + k + " строк(и)";
         for (j = 0; j < m; j++)
         {
             ptr[n - k, j] = 0;
             for (i = 0; i < n - k; i++)
             {
                 ptr[n-k, j] += ptr[i, j];
             }
         }

            dataGridView3.AutoResizeColumns();
            dataGridView3.ColumnCount = n;
            for (i = 0; i <= n-k; i++)
            {
                dataGridView3.Rows.Add();
                for (j = 0; j < m; j++)
                {
                   dataGridView3.Rows[i].Cells[j].Value = ptr[i, j];
                }
            }


        }
```

```
        private void button4_Click(object sender, EventArgs e)
        {
            dataGridView2.Rows.Clear();
            dataGridView2.Columns.Clear();
            dataGridView3.Rows.Clear();
            dataGridView3.Columns.Clear();
            button3.Enabled = true;
            label9.Text = "";

            }
```

26. По окончании проектирования файл *LR_1.cpp* будет выглядеть так:

```
using System;
using System.Collections.Generic;
using System.ComponentModel;
using System.Data;
using System.Drawing;
using System.Linq;
using System.Text;
using System.Threading.Tasks;
using System.Windows.Forms;
using System.Windows.Forms.DataVisualization.Charting;
namespace massiv
{
    public partial class Form1 : Form
    {

        public Form1()
        {
          InitializeComponent();

        }


        int i = 0;
        private void button1_Click(object sender, EventArgs e)
        {
           button2.Enabled = false;
           label8.Text = "";
           if (numericUpDown4.Value < numericUpDown5.Value)
           { label8.Text = "Макс значение не м.б. меньше мин
значения!"; return; }
           int count, current = 0;
           count = (Convert.ToInt32(numericUpDown2.Value) -
Convert.ToInt32(numericUpDown1.Value)) /
Convert.ToInt32(numericUpDown3.Value) + 1;
           for (int n = Convert.ToInt32(numericUpDown1.Value); n <=
Convert.ToInt32(numericUpDown2.Value); n +=
Convert.ToInt32(numericUpDown3.Value))
           {
               int[] vptr=new int[n];
```

```
                Random rand = new Random();
                for(int j=0;j<n;j++)
                  {
                    vptr[j] =
rand.Next(Convert.ToInt32(numericUpDown5.Value),
Convert.ToInt32(numericUpDown4.Value));
                  }
                if(checkBox1.Checked)
                 {
                   dataGridView1.ColumnCount = n+1;
                   dataGridView1.Rows.Add();
                   dataGridView1.Rows[i].Cells[0].Value = "Исходный
массив";
                   for (int j = 0; j < n; j++)
                          { dataGridView1.Rows[i].Cells[j + 1].Value =
vptr[j]; }
                   i++;
                 }
                sort(vptr,n);
                current += 1;
                  progressBar1.Value = 100 * current / count;
            }
        }
          private void sort(int[] p, int n)
          {
           int k = 0, sr = 0, obm = 0, m = 0;
              for (int j = 0; j < n; j++)
              {
                  if (p[j] == 0) k++;
                  else p[j - k] = p[j];
              }
              n -= k;
              sr += n;
              if (n == 0) { label8.Text = "В массиве одни нули"; return;
}
              for (m = 0; m < n - 1; m++)
                  for (int j = m + 1; j < n; j++)
                  {
                      if (p[m] > 0 && p[j] > 0 && p[m] < p[j]) {
swap(ref p[m], ref p[j]); obm++; }
                      if (p[m] < 0 && p[j] < 0 && p[m] > p[j]) {
swap(ref p[m], ref p[j]); obm++; }
                      sr += 6;
                  }
              if (checkBox1.Checked)
              {
                  dataGridView1.AutoResizeColumns();
                  dataGridView1.Rows.Add();
                  dataGridView1.Rows[i].Cells[0].Value = "Получен
массив";
                  for (int j = 0; j < n; j++)
```

```
                { dataGridView1.Rows[i].Cells[j + 1].Value = p[j]; }
                i++;
            }
            if (Convert.ToInt32(numericUpDown1.Value) ==
Convert.ToInt32(numericUpDown2.Value))
            { label8.Text = "Количество сравнений=" +
Convert.ToString(sr) + " Количество обменов=" + Convert.ToString(obm);
}
            if (checkBox2.Checked)
            {
                chart1.Series[0].Points.AddXY(n, sr);
                chart2.Series[0].Points.AddXY(n, obm);
            }
         }


        void swap(ref int x, ref int y)
        { int z = x; x = y; y = z; }



        private void button2_Click(object sender, EventArgs e)
        {
            chart1.Series[0].Points.Clear();
            chart2.Series[0].Points.Clear();
            dataGridView1.Rows.Clear();
            dataGridView1.Columns.Clear();
            i = 0;
            button2.Enabled = true;
        }
        int n, m;
        private void button3_Click(object sender, EventArgs e)
        {
            int i, j, k, q;
            button3.Enabled = false;
            if (numericUpDown8.Value < numericUpDown9.Value)
            { label9.Text = "Макс значение не м.б. меньше мин
значения!"; return; }
            n = Convert.ToInt32(numericUpDown6.Value);
            m = Convert.ToInt32(numericUpDown6.Value);
            int[,] ptr;
            ptr = new int[m, n];
            Random rand = new Random();
            dataGridView2.AutoResizeColumns();
            dataGridView2.ColumnCount = n;
            for (i = 0; i < n; i++)
            {
                dataGridView2.Rows.Add();
                for (j = 0; j < m; j++)
                {
```

```
                    ptr[i,j] =
rand.Next(Convert.ToInt32(numericUpDown9.Value),
Convert.ToInt32(numericUpDown8.Value));
                    dataGridView2.Rows[i].Cells[j].Value = ptr[i,j];
                }
            }
            q = 0;
            k = 0;
           if (ptr[n - 1, m - 1] < 0) k++;
         for(q=0;q<n-1;q++)

            {
                if (ptr[q, m - 1] < 0)
                {
                    k++;
                    for (i = q; i < n - 1; i++)
                    {
                        for (j = 0; j < m; j++) ptr[i, j] = ptr[i + 1,
j];

                    }
                    q--;
                }
                if (k + q+1 == n) { break; }
            }

         if (k == n) { label9.Text = "В матрице удалены все строки";
return; }
         if (k == 0) { label9.Text = "В матрице нет удаленных строк";
return; }
         label9.Text = "В матрице удалено " + k + " строк(и)";
         for (j = 0; j < m; j++)
         {
             ptr[n - k, j] = 0;
             for (i = 0; i < n - k; i++)
             {
                 ptr[n-k, j] += ptr[i, j];
             }
         }

            dataGridView3.AutoResizeColumns();
            dataGridView3.ColumnCount = n;
            for (i = 0; i <= n-k; i++)
            {
                dataGridView3.Rows.Add();
                for (j = 0; j < m; j++)
                {
                   dataGridView3.Rows[i].Cells[j].Value = ptr[i, j];
                }
            }
```

```
        }


        private void button4_Click(object sender, EventArgs e)
        {
            dataGridView2.Rows.Clear();
            dataGridView2.Columns.Clear();
            dataGridView3.Rows.Clear();
            dataGridView3.Columns.Clear();
            button3.Enabled = true;
            label9.Text = "";
        }

    }
}
```

Тестирование и использование приложения

1. Запустите приложение на выполнение, нажав быстрые кнопки *Сохранить все* и *Запуск*.

2. Подготовьте приложение к тестированию задания 1, щелкнув на закладке *массив*, а затем *тестирование и использование* (рис.2.1). Включите индикатор *вывод в таблицу*.

3. Пользуясь *ПУСК* и *СБРОС*, убедитесь в работоспособности приложения с параметрами массива, заданными по умолчанию (рис.2.4).

4. Изменяя диапазон значений элементов массива, убедитесь в работоспособности приложения в случаях: а) максимальное значение меньше минимального, б) в массиве только нули, в) в массиве только положительные элементы, г) в массиве только отрицательные элементы, д) массив состоит из равных по величине элементов (положительных и отрицательных).

5. Перейдите к использованию приложения для построения графиков зависимостей затрат машинного времени от размера массива при разных диапазонах значений элементов массива. Включите индикатор *графики*. При заданных по умолчанию остальных параметрах задайте конечный размер массива – 1000 и нажмите *ПУСК*. По окончании обработки массива получите на вкладе *графики* результат, представленный на рис.2.5.

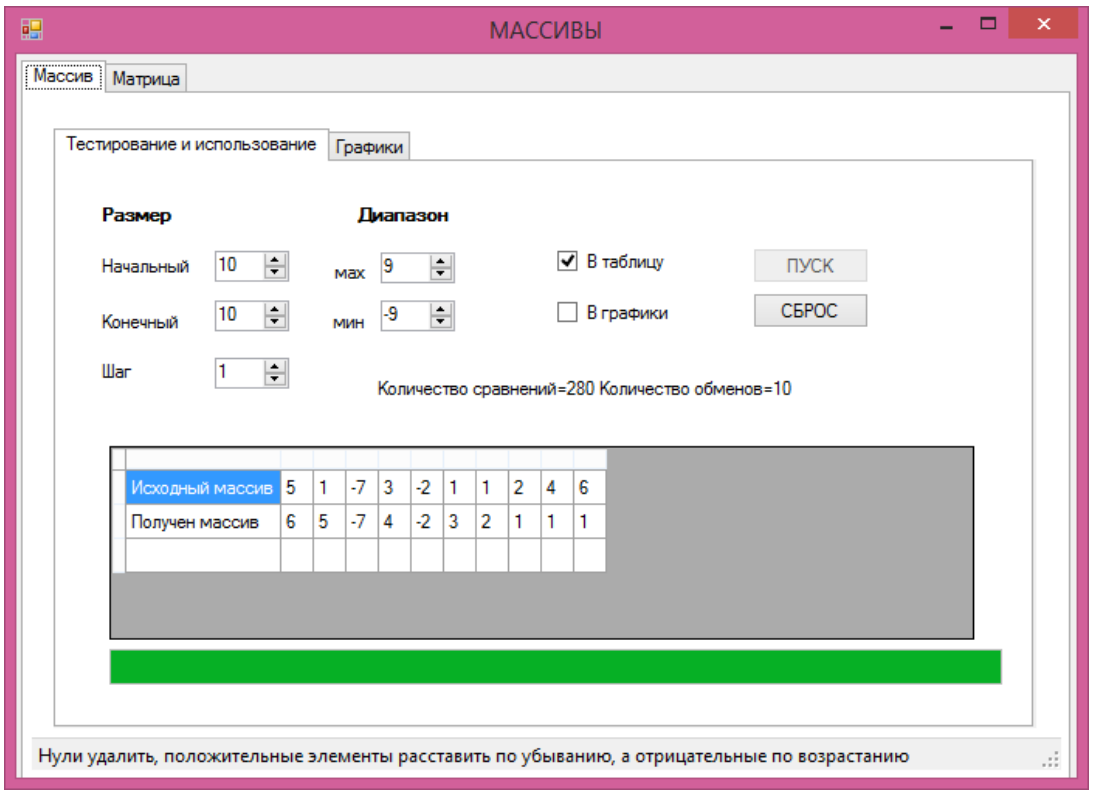

Рисунок 2.4 – результаты тестирования по заданию 1

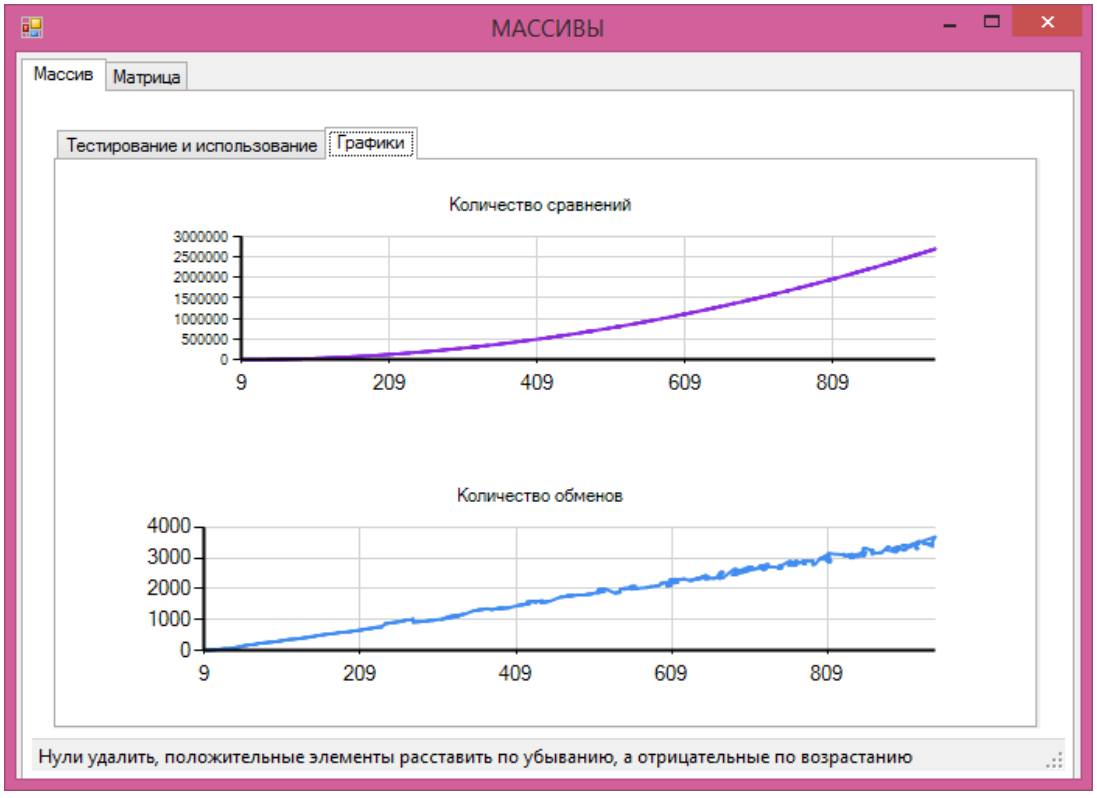

Рисунок 2.5 – зависимости затрат машинного времени от размера массива

6. Установите влияние диапазона значений элементов и размера массива на ход указанных зависимостей. В частности, установите параметры диапазона такими, чтобы количество обменов было нулевым при любом размере массива.

7. Подготовьте приложение к тестированию и выполнению задания 2, щелкнув на закладке *матрица* (рис.2.3).

8. Протестируйте приложение для заданных по умолчанию параметрах матрицы, щелкая на кнопках *ПУСК* и *СБРОС* (рис.2.6).

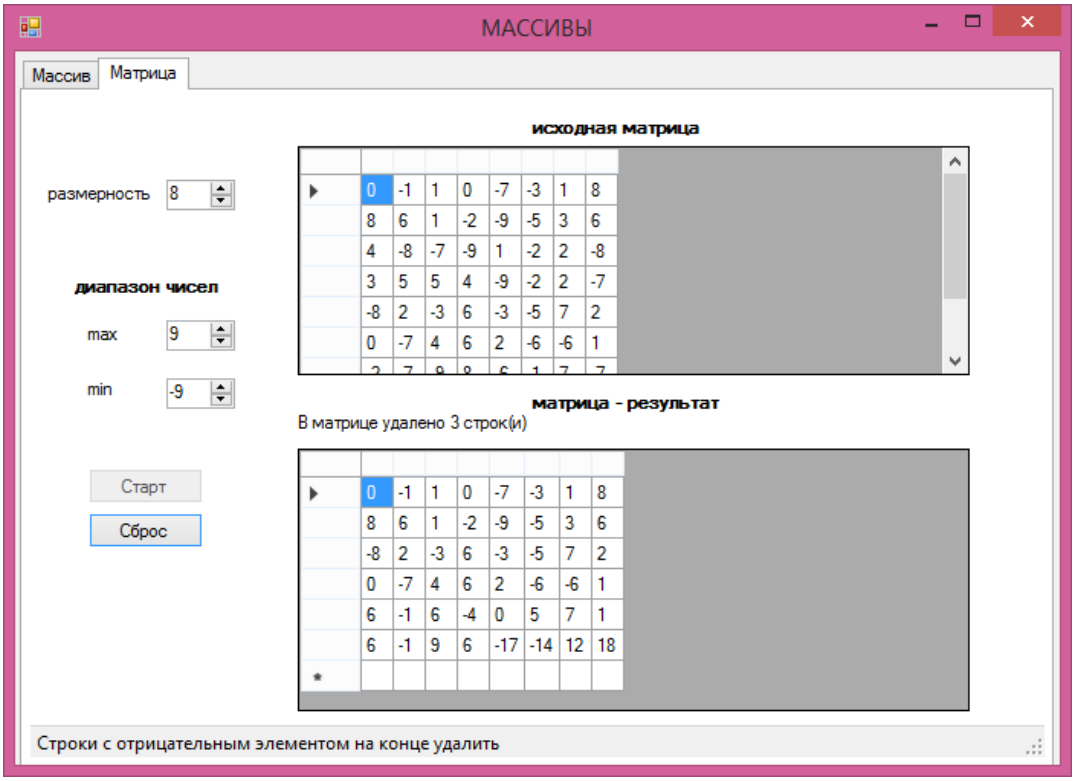

Рисунок 2.6 – результаты тестирования по заданию 2

9. Изменяя диапазон значений элементов матрицы, убедитесь в работоспособности приложения в случаях: а) максимальное значение меньше минимального, б) удалены все строки, в) нет удаленных строк. Установите влияние диапазона на среднее количество удаленных строк.

10. Проделайте то же самое, варьируя размерами матрицы.

11. Для завершения работы щелкните на кнопке формы “*Закрыть*” и выйдите из среды *VS*.

Контрольные вопросы

1. Как еще можно использовать свойство **Text** (надпись) формы? Приведите примеры фрагментов кода.

2. Какие свойства компонента **tabControl** используются в приложении? Как установить значения этих свойств?

3. Поясните назначения компонентов **dataGridView** в приложении. Значения каких свойств использованы по умолчанию, а у каких значения по умолчанию пришлось изменить?

4. Расскажите порядок установки значений свойств компонента **chart**. Каким свойствам можно задать другие значения без ущерба для качества представления результатов?

5. Какую функцию выполняют в приложении компоненты **checkBox**? Для ответа воспользуйтесь кодом.
6. Как задаются размеры массива и матрицы? Поясните по коду.
7. Используя код, объясните, как задается диапазон значений элементов массива.
8. Используя код, объясните, как задается диапазон значений элементов матрицы.
9. Используя код, объясните, как задаются значения элементов массива и матрицы.
10. Поясните назначение компонентов **statusStrip**. Укажите и объясните фрагменты кода, относящиеся к этим компонентам. Как вывести сообщение в компоненты во время выполнения приложения?
11. Поясните назначение компонента **pogressBar**. Укажите и объясните фрагменты кода, относящиеся к этому компоненту. Изобразите закон изменения переменной *current* по времени.
12. Используя код, объясните, как удаляются из массива нули. Представьте алгоритм.
13. Используя код, объясните, как сортируются в массиве положительные элементы. Представьте алгоритм.
14. Используя код, объясните, как сортируются в массиве отрицательные элементы. Представьте алгоритм.
15. Как выводятся на экран исходный массив и массив, полученный в результате сортировки? Для ответа используйте код.
16. Как строятся графики зависимостей? Для объяснения воспользуйтесь кодом. Объясните вид зависимостей.
17. Как выводятся на экран исходная матрица и матрица, полученная после обработки? Для ответа обратитесь к коду.
18. Используя код, объясните, как выделяется динамическая память под формируемые массив и матрицу и как она освобождается. Представьте алгоритмы.
19. Какие операции с динамической памятью выполняются во время обработки матрицы?
20. Представьте алгоритм удаления из матрицы строк с последними отрицательными элементами.
21. Представьте алгоритм добавления строки в матрицу согласно коду.
22. Что происходит при щелчке на кнопках *ПУСК* и *ПУСК* в первой вкладке?
23. Что происходит при щелчке на кнопках *СБРОС* и *СБРОС* во второй вкладке?

Задания

1. В матрице удалить строки, содержащие нули, а затем добавить строку, элементы которой равны произведениям элементов в соответствующих столбцах.

2. В матрице удалить столбцы с нулевыми элементами ниже главной диагонали, а затем добавить столбец, элементы которого равны суммам элементов в соответствующих строках.
3. В матрице удалить столбцы, в которых количество отрицательных элементов превышает заданное, а затем в качестве первого добавить столбец с максимальными элементами по строкам.
4. В матрице удалить строки с нулевыми элементами выше главной диагонали, а затем в качестве третьей добавить строку, элементы которой равны разностям соответствующих элементов первой и второй строк
5. В матрице удалить столбцы с положительными суммами элементов, а затем в качестве первого вставить столбец из минимальных элементов соответствующих строк.
6. В матрице удалить строку с минимальным произведением элементов, а затем в качестве второй добавить строку, элементы которой равны разностям элементов первой и последней строк.
7. В матрице удалить строки, последние элементы которых отрицательны, а затем в качестве первой добавить строку из элементов заданного массива.
8. В матрице удалить первую и последнюю строки, а затем добавить строку из максимальных элементов соответствующих столбцов.
9. В матрице удалить столбцы с отрицательной суммой элементов, а затем добавить столбец из минимальных элементов соответствующих строк.
10. В матрице удалить столбцы с максимальным и минимальным элементами матрицы, а затем на место первого добавить столбец из произведений элементов соответствующих строк.
11. В матрице удалить строки с элементами на главной диагонали, превышающими заданную величину, а затем в качестве первой вставить строку из максимальных элементов соответствующих столбцов.
12. В матрице удалить столбцы с положительными третьими элементами, а затем добавить столбец из элементов заданного массива.
13. Вычислить m-норму матрицы.
14. Вычислить l-норму матрицы.
15. Вычислить k-норму матрицы.
16. В матрице удалить строки с положительными последними элементами, а затем добавить строку из минимальных элементов по соответствующим столбцам.
17. Сформировать массивы из элементов в седловых точках матриц. *Примечание:* в седловой точке элемент является минимальным в строке и максимальным в столбце.
18. Приняв за характеристику столбца матрицы сумму модулей его отрицательных нечетных элементов, расположить столбцы матриц в соответствии с ростом характеристик.
19. Выполнить операции сглаживания матриц. Операция сглаживания дает матрицу того же размера, каждый элемент которой получается как среднее арифметическое соседей соответствующего элемента матрицы. Соседями

элемента a_{ij} в матрице называют элементы a_{kl} с $i-1 \le k \le i+1, j-1 \le l \le j+1, (k,l) \ne (i,j)$.

20. Элемент матрицы называется локальным минимумом, если он строго меньше всех имеющихся у него соседей. Сформировать массив из количеств локальных минимумов обработанных матриц.

21. Осуществить циклический сдвиг элементов прямоугольных матриц на *n* элементов вправо или вниз (в зависимости от указанного режима); *n* может быть больше количества элементов в строке или столбце матрицы.

22. Осуществить циклический сдвиг элементов квадратных матриц вправо на *k* элементов таким образом: элементы первой строки сдвигаются в последний столбец сверху вниз, из него – в последнюю строку справа налево, из неё - в первый столбец снизу вверх, из него – в первую строку; для остальных элементов – аналогично.

23. Сортировать матрицы следующим образом: помещать элементы на диагонали, начиная с главной диагонали, в порядке убывания.

24. В матрице удалить строку с минимальным элементом матрицы, а затем добавить отсортированную по возрастанию предыдущую строку.

25. В матрицах проверять указанную строку на ортогональность остальным строкам.

26. В матрице удалить строку с максимальным по модулю элементом матрицы, а затем в качестве первой добавить строку, элементы которой равны суммам модулей элементов в соответствующих столбцах.

27. Проверять матрицы на попарную ортогональность строк; обработку каждой матрицы завершать выводом списка попарно ортогональных строк.

28. Характеристикой строки матрицы назовём сумму её отрицательных четных элементов. Расположить строки в соответствии с убыванием характеристик.

29. В матрице удалить столбец с максимальным элементом матрицы, а затем вставить заданный столбец перед столбцом с минимальным элементом полученной матрицы.

30. В матрицу добавить строку из элементов заданного массива, а затем удалить строки с положительной суммой элементов.