Государственное бюджетное профессиональное образовательное учреждение Астраханской области «Астраханский колледж вычислительной техники»

К О М П Л Е К С

МЕТОДИЧЕСКИХ РЕКОМЕНДАЦИЙ
по проведению лабораторных работ

по МДК.03.02 *Инструментальные средства разработки программного обеспечения*

по специальности *09.02.03*

Программирование

в компьютерных системах

АКВТ.09.02.03.ЛР46.0000

Листов: *58*

2018

ПРЕДИСЛОВИЕ

Методические указания предназначены для реализации государственных требований к минимуму содержания и уровню подготовки выпускников по специальности 09.02.03 Программирование в компьютерных системах при очной форме обучения, по МДК 03.02 Инструментальные средства разработки программного обеспечения, в условиях действия государственного образовательного стандарта среднего профессионального образования.

Созданный методический комплекс по лабораторным работам направлен на:

– Обобщение, систематизацию, углубление теоретических знаний по следующим темам МДК 03.02 Инструментальные средства разработки программного обеспечения: «Использование технологии LINQ. Запросы к массиву данных и коллекции», «Использование технологии LINQ. Работа с XML-документами», «Программирование Web-ориентированных приложений. Проверка введённых данных в Web-форму», «Программирование Web-ориентированных приложений. Организация передачи данных между Web-страницами», «Типичные Web-ориентированные приложения ASP.NET», «Создание Web-служб и их клиентов. Создание службы и клиента», «Использование технологии WPF. Простейшие приложения», «Использование технологии WPF. Использование эффектов анимации».

– Формирование умений применять полученные теоретические знания в практической деятельности.

– Развитие аналитических, проектировочных, конструктивных умений.

– Выработку самостоятельности, ответственности, точности и творческой инициативы.

В процессе проведения лабораторных работ студенты выполняют одну или несколько лабораторных работ (заданий) в соответствии с изучаемым содержанием учебного материала по «МДК 03.02 Инструментальные средства разработки программного обеспечения». В ходе выполнения лабораторных работ у студентов формируются практические умения и навыки обращения с различным лабораторным оборудованием, а также исследовательские умения (создавать, форматировать информацию, используя прикладные программы, наблюдать, сравнивать, анализировать, устанавливать зависимости, делать выводы и обобщения, самостоятельно вести исследования, оформлять результаты).

Содержание и количество часов лабораторных работ определено в рабочей программе данного междисциплинарного курса. Лабораторные работы как вид учебного занятия проводиться в специально оборудованных лабораториях – ВЦ.

Необходимые структурные элементы лабораторной работы:

1) инструктаж, проводимый преподавателем;
2) самостоятельная деятельность студентов;
3) обсуждение итогов выполнения лабораторной работы.

Перед выполнением лабораторной работы проводится проверка знаний студентов – их теоретической готовности к выполнению задания, в виде вводного контроля.

Лабораторные работы, представленные в данном комплексе, носят и репродуктивный, и частично-поисковый, и поисковый характер. При групповой форме организации занятий одна и та же работа выполняется бригадами по 2 – 3 человека. Результаты выполнения лабораторных работ оформляются студентами в виде отчета.

ПРАВИЛА ПРОВЕДЕНИЯ ЛАБОРАТОРНЫХ РАБОТ

Перед выполнением лабораторных работ по МДК студенты должны:

– строго выполнить весь объем домашней подготовки, т.е. ознакомиться с теоретическим материалом, указанный в описаниях соответствующих лабораторных работ;

– знать, что выполнению каждой работы предшествует проверка готовности студента, которая производиться преподавателем;

– знать, что после выполнения работы группа студентов, которая назначается преподавателем на весь период работы, должна представить отчёт о проделанной работе с обсуждением полученных результатов и выводов;

– ознакомиться с требованиями и процедурами выставления окончательной оценки (зачёта) по работе и порядок выполнения пропущенных работ по уважительным и неуважительным причинам.

1 «Создание LINQ-запрос к массиву данных»

1 Цель работы

1.1 Изучить технологию LINQ.
1.2 Научиться создавать LINQ-запросы к массиву данных.

2 Приборы и оборудование:

2.1 Методические указания.
2.2 ПЭВМ типа IBM PC/XT.
2.3 MS Office.
2.4 Visual C#

3 Порядок выполнения работы:

LINQ-запрос представляет собой выражение, извлекающее данные из источника данных. Все операции запроса LINQ состоят из трех различных действий: получение источника (в нашем случае — это присвоение начальных значений исходному массиву) данных, создание запроса (начинается с предложения from) и непосредственное выполнение запроса (обычно это цикл foreach). Рассмотрим две задачи, первая — из массива имен извлекаем имена длиной шесть символов, записывая их или в список (коллекцию), или в новый массив. Другая задача — из массива целых чисел выбираем только те, значения которых больше четырех, также записывая результат запроса в список или массив.

3.1 Запустим Visual Studio, закажем новый проект шаблона Windows Forms Application C#. 3.2 Из панели элементов перенесем в форму текстовое поле. Далее через щелчок правой кнопкой мыши перейдем к вкладке программного кода (листинг 1).

Листинг 1 Извлечение данных из массивов

```
// Решаем две задачи по выбору элементов из массива с помощью
стандартных
// запросов технологии LINQ
using System;
using System.Linq;
using System.Windows.Forms;
// Другие директивы using удалены, поскольку они не используются
// в данной программе
namespace Linq1
{
public partial class Form1 : Form
{
public Form1()
{
InitializeComponent();
```

```
this.Text = "Технология LINQ"; textBox1.Multiline = true;
// ЗАДАЧА 1.
// Из массива имен выбрать имена с количеством букв, равным шести,
// вывести эти имена в текстовое поле TextBox в алфавитном порядке,
// при этом все буквы перевести в верхний регистр.
// Решение:
string СтрокаИмен = "Витя Лариса Лариса Лена Андрей Женя " +
"Александр Лариса Виктор Света Оксана Наташа";
// Из строки имен получаем массив имен, задавая в качестве
// разделителя подстрок символ пробела:
string[] Имена = СтрокаИмен.Split(' ');
// или проще: string[] Имена =
// { "Витя", "Лариса", "Лариса", "Лена", "Андрей", "Женя",
// "Александр", "Лариса", "Виктор", "Света", "Оксана", "Наташа" };
textBox1.Text = "ЗАДАЧА 1. В списке имен:\r\n\r\n";
foreach (string x in Имена)
textBox1.Text = textBox1.Text + x + " ";
// В результате LINQ-запроса получаем список имен с количеством букв,
// равным шести:
var Запрос = from s in Имена
where s.Length == 6 // - условие выбора
orderby s // - сортировать в алфавитном порядке
select s.ToUpper(); // - перевод в верхний регистр
// s - переменная диапазона, схожа с переменной итерации в foreach
// Удаляем элементы-дубликаты из списка имен:
Запрос = Запрос.Distinct();
// Или таким образом:
// Запрос = Запрос.Union(Запрос);
textBox1.Text = textBox1.Text + "\r\n\r\n" +
"выбираем имена с количеством букв равным шести, при " +
"этом избавляемся от дублирования имен:\r\n\r\n";
foreach (string x in Запрос) // x - переменная итерации
// в цикле foreach
textBox1.Text = textBox1.Text + x + " ";
textBox1.Text = textBox1.Text + "\r\n\r\n";
//~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
~~~~~~~~~~~~
// ЗАДАЧА 2. Из массива целых чисел X[] требуется выбрать числа,
// значения которых >= 4, и записать эти числа в список Y,
// отсортировав выбранные числа по возрастанию.
// Решение.
// Инициализация массива целых чисел:
int[] X = { -2, 5, -23, 0, 7, -10, 11, 11, 14, 3, 8, 5, -5, 27, 8 };
textBox1.Text += "ЗАДАЧА 2. Из заданного массива X:\r\n\r\n";
foreach (int x in X)
textBox1.Text = textBox1.Text + x + " ";
```

```
textBox1.Text = textBox1.Text + "\r\n\r\nвыбираем числа, значения " +
"которых >= 4 и записываем их в список (коллекцию) Y, " +
"исключая элементы-дубликаты:\r\n\r\n";
// Y - это список, в который помещаются выбранные элементы:
var Y = from x in X
where x >= 4 // - условие выбора
orderby x // - сортируем по возрастанию
select x;
// Таким образом можно получить результат запроса в массив:
// int[] Y = (from x in X // Здесь Y - это уже массив;
// where x >= 4 // - условие выбора;
// orderby x // - сортируем по возрастанию;
// select x).ToArray(); // - преобразование списка в массив
// Удаляем элементы-дубликаты из списка целых чисел:
var Z = Y.Distinct();
// Или таким образом:
// var Z = Y.Union(Y);
// Вывод элементов списка Y в метку textBox1:
foreach (var z in Z)
textBox1.Text = textBox1.Text + z.ToString() + " ";
}
}
}
```

Как видно из программного кода, после присвоения массиву Имена начальных значений создаем запрос, который предусматривает выбор (select) из (from) массива Имена строк длиной (Length) ровно шесть символов (условие where). Запись выбранных имен выполняется в список Запрос с сортировкой списка в алфавитном порядке имен. Далее для удаления повторяющихся имен в списке используем функцию Distinct. В комментарии показано, как можно для этой же цели использовать функцию Union, которая, вообще говоря, находит объединение двух множеств, т. е. выбирает в двух списках одинаковые элементы с удалением повторяющихся элементов. Поскольку мы объединяем два одинаковых списка, то получаем просто удаление повторяющихся элементов. Цикл foreach выводит список Запрос в текстовое поле textBox1.

Следующая задача, решенная в данном программном коде, аналогична. Задан массив целых чисел X . Из этого массива выбираем в список Y элементы массива, значения которых больше или равны четырем. В комментарии показано, как можно записать результаты LINQ-запроса в массив. Удаление повторяющихся элементов в списке выполнено также с использованием функции Distinct.

Убедиться в работоспособности программы можно, открыв решение Linq1.sln папки Linq1.

Задачи для самостоятельного выполнения:

1. Из массива выбрать числа больше 5 и меньше 10.
2. Подсчитать сумму элементов массива больше 5.
3. Определить количество фамилий, длиной больше 4 символов.

4. Вывести четные элементы массива. Для определения четности использовать символ %.

5. Найти среднее арифметическое нечетных элементов массива.

4 Содержание отчёта:

4.1 Тема работы.
4.2 Цель работы.
4.3 Приборы и оборудование.
4.4 Порядок выполнения работы.
4.5 Выводы.

2 Создание LINQ-запросов к коллекции данных

1 Цель работы

1.1 Изучить технологию LINQ.
1.2 Научиться создавать LINQ-запросы к коллекции данных.

2 Приборы и оборудование:

2.1 Методические указания.
2.2 ПЭВМ типа IBM PC/XT.
2.3 MS Office.
2.4 Visual C#

3 Порядок выполнения работы:

В некоторых случаях хранение данных в коллекции (скажем, в списке типа **List)** может оказаться более эффективным, чем в массиве. Например, если число элементов в массиве при работе изменяется часто или нельзя предсказать максимальное количество необходимых элементов, то можно получить большую производительность при использовании коллекции. Но если размер не изменяется или изменяется довольно редко, то массив, пожалуй, более эффективен.

3.1 Запустить Visual Studio.
3.2 Создать новый проект шаблона Windows Forms Application C#.
3.3 Из панели элементов перенесем в форму текстовое поле. Далее через щелчок правой кнопкой мыши перейдем к вкладке программного кода (листинг 1).

Листинг 1

```
// Решаем три различных задачи по выбору элементов (объектов) из списка с помощью
// стандартных запросов технологии LINQ
using System;
using System.Collections.Generic;
using System.Linq;
using System.Windows.Forms;
// Другие директивы using удалены, поскольку они не используются в данной программе
namespace Linq2
{
  public partial class Form1 : Form
  {
    public Form1()
    {
      InitializeComponent();
      this.Text = "Технология LINQ"; textBox1.Multiline = true;
```

```
            // ЗАДАЧА 1. Из списка строк выбрать нужные записи задав
условие выбора
            textBox1.Text = "ЗАДАЧА 1: Из списка имен:\r\n";
            // Объявление списка строк и его заполнение:
            var Список = new List<string> { "Витя", "Света", "Андрей",
                                            "Лариса", "Маша", "Наташа" };
            // или var Список = new List<string>(); // - список строк
            // Список.Add("Витя"); Список.Add("Света");
Список.Add("Андрей");
            // Список.Add("Лариса"); Список.Add("Маша");
Список.Add("Наташа");
            // Некоторые манипуляции со списком:
            int n = Список.Count; // Кол-во элементов в списке:
            // Получение из списка его элемента с индексом 3 (как в массиве):
            string A = Список.ElementAt<string>(4);
            Boolean Ответ = Список.Remove("Лариса"); // - удаление из списка
            // Преобразовать список в строковый массив:
            string[] МассивСтрок = Список.ToArray();
            foreach (var x in Список)
               textBox1.Text = textBox1.Text + x.ToString() + " ";
            textBox1.Text += "\r\n выбираем имена длиной четыре символа:\r\n";
            // СписокВыбранныхИмен - это новый список, куда попадают
выбранные
            // строки в результате LINQ-запроса:
            var СписокВыбранныхИмен = from Имя in Список
                                      where Имя.Length == 4 // - условие выбора
                                      orderby Имя // - сортировать список Z
                                      select Имя;
            // Вывод списка выбранных имен в текстовое поле textBox1:
            foreach (var x in СписокВыбранныхИмен)
               textBox1.Text = textBox1.Text + x.ToString() + " ";

//~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
            // ЗАДАЧА 2.
            textBox1.Text +=
               "\r\n\r\nЗАДАЧА 2: Из списка сотрудников предприятия " +
                        "выбираем не курящих для повышения зарплаты:\r\n\r\n";
            // Заполняем список сотрудников:
            List<Сотрудник> Сотрудники = new List<Сотрудник>{
              new Сотрудник {Имя="Карапузова Ирина", Возраст=27,
КуритЛи=true },
              new Сотрудник {Имя="Зиборов Виктор", Возраст=47,
КуритЛи=false },
              new Сотрудник {Имя="Ломачинская Светлана", Возраст=34,
                                                КуритЛи=false },
```

```
                    new Сотрудник {Имя="Стороженко Светлана", Возраст=34,
КуритЛи=false },
                    new Сотрудник {Имя="Еременко Татьяна", Возраст=34,
КуритЛи=true },
                    new Сотрудник {Имя="Погребицкий Олег", Возраст=42,
КуритЛи=true },
                                                    };
                var СписокНекурящихСотрудников = from Сотрудник in
Сотрудники
                                                 where Сотрудник.КуритЛи == false
                                                 orderby Сотрудник.Имя
                                                 select Сотрудник;
                // Вывод списка некурящих сотрудников в текстовое поле textBox1:
                foreach (var x in СписокНекурящихСотрудников)
                   textBox1.Text = textBox1.Text + string.Format("{0} - возраст " +
                                                 "- {1}\r\n", x.Имя, x.Возраст);

//~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
                // ЗАДАЧА 3.
                textBox1.Text += "\r\nЗАДАЧА 3: Из списка студентов факультета "
+
                                                 "выбираем двоечников:\r\n\r\n";
                // Каждый элемент в списке содержит фамилию студента и
полученные
                // им текущие оценки:
                List<Студент> Студенты = new List<Студент>
            {
              new Студент {Фамилия="Зиборов", Оценки= new List<int> {5, 4, 4,
5}},
              new Студент {Фамилия="Стороженко", Оценки= new List<int> {3, 3,
2, 3}},
              new Студент {Фамилия="Ломачинская", Оценки= new List<int> {3,
4, 4, 5}},
              new Студент {Фамилия="Погребицкий", Оценки= new List<int> {2,
4, 3, 2}},
              new Студент {Фамилия="Левочкин", Оценки= new List<int> {3, 3, 4,
3}}
            };
              // Для доступа к внутреннему списку оценок предложение From
              // используем два раза:
              var СписокДвоечников = from Студент in Студенты
                                     from Оценка in Студент.Оценки
                                     where Оценка <= 2
                                     orderby Студент.Фамилия
                                     select new { Студент.Фамилия, Оценка };
              foreach (var Студик in СписокДвоечников)
```

```
                    textBox1.Text += string.Format("Студент {0} " +
                            "имеет оценку: {1}\r\n", Студик.Фамилия, Студик.Оценка);
                // Строка со студентом Погребицким выводится два раза, поскольку
он
                // имеет две двойки
            }

            private void textBox1_TextChanged(object sender, EventArgs e)
            {

            }
        }
        // Объявляем класс, содержащий имя сотрудника, его возраст, а также
        // информацию, курит ли он:
        public class Сотрудник
        {
            public string Имя { get; set; }
            public int Возраст { get; set; }
            public bool КуритЛи { get; set; }
        }
        // Объявляем класс, содержащий фамилию студента и список
полученных им оценок:
        public class Студент
        {
            public string Фамилия { get; set; }
            public List<int> Оценки { get; set; }
        }
    }
```

Первая задача довольно простая и очень похожа на задачи, которые мы решали с помощью технологии LINQ в предыдущем работе. Здесь, в этой новой задаче вместо массива имен требуется создать список имен, а далее из этого списка выбрать имена длиной четыре символа. Решение этой задачи, построение LINQ- запроса аналогично решению задачи из предыдущей работы, отличие состоит лишь в применении синтаксиса манипуляций со списком типа **List,** а не с массивом. Здесь также приведены некоторые важные техники для манипуляции списком, в частности получение элемента списка по указанному индексу (аналогия с массивом), удаление элемента, преобразование списка в строковый массив.

Вторая задача заключается в том, чтобы создать список сотрудников предприятия и из этого списка выбрать некоторых сотрудников по какому-либо признаку, например тех, кто не курит. При создании списка объявлен класс **Сотрудник,** который содержит три свойства: **Имя, возраст** и булеву переменную **КуритЛи.** В начале решения заполняем список сотрудников, а затем строим

LINQ-запрос для заполнения списка некурящих сотрудников. Этот список выводим в текстовое поле **textBox1,** используя цикл **foreach.**

Третья задача немного сложнее. Требуется создать список студентов факультета, содержащий фамилию студента и список полученных им текущих оценок, т. е. список оценок должен быть "вложен" в список студентов. Из списка студентов необходимо выбрать тех, кто имеет в списке своих оценок хотя бы одну двойку. Для решения этой задачи вначале объявляем новый класс **Студент,** который имеет в качестве свойств класса фамилию студента и список (типа **List)** оценок. Далее в начале решения третьей задачи мы привели синтаксис заполнения списка студентов. Затем строим LINQ-запрос, где, поскольку нам необходимо искать элемент списка внутри "вложенного" списка, мы используем предложение **From** два раза. Поскольку студент Погребицкий имеет две двойки в списке своих текущих оценок, он в списке двоечников фигурирует дважды.

Задачи для самостоятельного выполнения:

1. Даны сведения о 10 сотрудниках: ФИО, должность, оклад. Выбрать сотрудников с окладом больше 10000
2. Даны сведения о 10 студентах: ФИО, группа, средний балл. Выбрать студентов, обучающихся в группе 31 и имеющие средний балл больше 4.5
3. Даны сведения о 10 клиентах: ФИО, статус, возраст. Выбрать клиентов со статусом VIP и возрастом в диапазоне от 20 до 30 лет
4. В банке у каждого клиента может быть несколько счетов с указанием их остатков в рублях. Вывести фамилии клиентов, у которых есть счета с остатком менее 1000 руб.
5. Имеется 5 групп. В каждой группе учится по 5 студентов. Вывести фамилии студентов с фамилией больше 6 символов

4 Содержание отчёта:

4.1 Тема работы.
4.2 Цель работы.
4.3 Приборы и оборудование.
4.4 Порядок выполнения работы.
4.5 Выводы.

3 Группировка элементов списка с помощью LINQ-запросов

1 Цель работы

1.1 Изучить технологию LINQ.
1.2 Научиться группировать элементы списков с помощью LINQ-запросов.

2 Приборы и оборудование:

2.1 Методические указания.
2.2 ПЭВМ типа IBM PC/XT.
2.3 MS Office.
2.4 Visual C#

3 Порядок выполнения работы:

В данном примере используем стандартный шаблон LINQ-запроса для группировки элементов списка. Программа формирует список продуктов питания. Следует организовать такие LINQ-запросы, которые разделят искомый список на две группы по критерию цены (больше или меньше 90 руб. за единицу) и вычислят среднюю цену по каждой группе продуктов.

3.1 запустить Visual Studio.
3.2 Выбрать проект шаблона **Windows Forms Application.**
3.3 Указать имя **Name** — LinqЦеныНаПродукты.
3.4 Попав в конструктор формы, из панели элементов **Toolbox** перетащим текстовое поле **TextBox** для вывода в него групп списка. В свойствах текстового поля разрешим ввод множества строк, для этого свойство Multiline переведем в состояние true. Затем на вкладке программного кода введем текст, представленный в листинге 1

Листинг 1

```
// Программа формирует список некоторых продуктов питания. Первый LINQ-запрос
// группирует элементы списка по критерию цены: в первом списке оказываются
// продукты, цена за единицу которых меньше или равна 90 руб., а во втором,
// соответственно, больше 90 руб. Второй LINQ-запрос вычисляет среднюю цену
// продукта по каждой группе. Результаты запросов выводятся в текстовое поле.
using System;
using System.Collections.Generic;
using System.Linq;
using System.Windows.Forms;
```

```
        // Другие директивы using удалены, поскольку они не используются в
данной программе
        namespace LinqЦеныНаПродукты
        {
          public partial class Form1 : Form
          {
            public Form1()
            {
              InitializeComponent();
              textBox1.Multiline = true;
              base.Text = "Группировка элементов списка с помощью LINQ-
запроса";
              // Заполняем список продуктов:
              List<Продукт> Продукты = new List<Продукт>{
                      new Продукт {Наименование="Творог",  Цена=112.50F },
                      new Продукт {Наименование="Хлеб",    Цена=18.75F },
                      new Продукт {Наименование="Печенье", Цена=93.75F },
                      new Продукт {Наименование="Чай",     Цена=76.25F },
                      new Продукт {Наименование="Мясо",    Цена=150.00F },
                      new Продукт {Наименование="Гречка",  Цена=62.50F },
                                          };
              var Запрос1 =
                       from П in Продукты
                       group П by new
                       {
                          Критерий = П.Цена > 90,
                       }
                          into g
                          select g;
              var Запрос2 =
                       from p in Продукты
                       group p by p.Цена > 90 into g
                       select new
                       {
                          g.Key,
        СредЦенаПоГруппе = g.Average(p => p.Цена)
                       };
              Single СредЦенаПоГруппе1 =
Запрос2.ElementAt(0).СредЦенаПоГруппе;
              Single СредЦенаПоГруппе2 =
Запрос2.ElementAt(1).СредЦенаПоГруппе;
              // Вывод результатов обоих запросов в текстовое поле:
              foreach (var Группа in Запрос1)
              {
                if (Группа.Key.Критерий == false)
                   textBox1.Text += "\r\nЦены 90 руб или меньше:";
```

```
                else
                textBox1.Text += "\r\nЦены больше 90 руб:";
                foreach (var Прод in Группа)
                {
                   textBox1.Text += String.Format("\r\n{0} - {1}",
                                    Прод.Наименование, Прод.Цена);
                }
                if (Группа.Key.Критерий == false)
                   textBox1.Text += String.Format(
                      "\r\nСредняя цена по данной группе = {0} руб.\r\n",
                                                СредЦенаПоГруппе2);
                else
                   textBox1.Text += String.Format(
                      "\r\nСредняя цена по данной группе = {0} руб.\r\n",
                                                СредЦенаПоГруппе1);
             }
          }

          private void textBox1_TextChanged(object sender, EventArgs e)
          {

          }
       }
       public class Продукт
       {
          public String Наименование { get; set; }
          public Single Цена { get; set; }
       }
    }
```

В начале программы формируем список продуктов питания. Для этого объявляем новый класс Продукт, в котором для каждого продукта задаем два поля: наименование продукта и его цену. После заполнения списка продуктов (для упрощения в этом списке всего шесть продуктов) задаем первый LINQ-запрос, обеспечивающий деление списка на две группы. Второй LINQ-запрос вычисляет среднюю цену продукта по каждой группе. Вывод результатов обоих запросов организуем с помощью двух вложенных циклов foreach.

Задачи для самостоятельного выполнения:

1. Имеются сведения о сотрудниках и их заработной плате. Вывести сотрудников с окладом меньше 10000 руб.
2. Имеются сведения о студентах: ФИО, группа, размер стипендии. Вывести средний размер стипендии по одной из группы
3. В организации имеется 3 отдела. В каждом отделе имеется от 3 до 5 сотрудников. Вывести список сотрудников и средний оклад по каждому отделу

4. На предприятии имеется 3 склада. На каждом складе имеются товары с указанием их стоимости. Сгруппировать товары по складам и вывести максимальную цену товара на каждом складе.

4 Содержание отчёта:

4.1 Тема работы.
4.2 Цель работы.
4.3 Приборы и оборудование.
4.4 Порядок выполнения работы.
4.5 Выводы.

4 Создание XML-документа методами пространств имен System.XML.Linq

1 Цель работы

1.1 Изучить технологию LINQ.
1.2 Научиться создавать XML-документы методами пространств имен System.XML.Linq.

2 Приборы и оборудование:

2.1 Методические указания.
2.2 ПЭВМ типа IBM PC/XT.
2.3 MS Office.
2.4 Visual C#

3 Порядок выполнения работы:

Кроме пространства имен System.xmi, содержащего классы для обработки XML-документов, в Visual C# имеем пространство имен System.xml.Linq, содержащее классы, которые позволяют легко и эффективно изменять документы XML, а также организовывать LINQ-запросы. В данном примере оформим сведения о наших повседневных телефонных контактах в XML-документ. Этот документ будет иметь интуитивно понятную структуру: имя контакта, домашний и мобильный телефоны. Создав такой XML-документ и получив соответствующий XML-файл, его очень удобно просмотреть в MS Excel в виде таблицы, содержащей три столбца: имя контакта, домашний телефон и мобильный. Попутно обсудим структуру XML-документа.

3.1 Запустить среду Visual Studio
3.2 Выбрать проект шаблона Console Application
3.3 Указать имя Name — LinqСоздатьXML-документ. Затем на вкладке программного кода введем текст, представленный в листинге 1.

Листинг 1

```
using System;
using System.Xml.Linq;
// Другие директивы using удалены, поскольку они не используются в данной программе
namespace LinqСоздатьXML_документ
{
   class Program
   {
      static void Main(string[] args)
      {
         Console.Title = "Корневой элемент XML-документа";
         // Создаем новый XML-документ:
         XDocument XMLдокумент = new XDocument(
```

```
                    // Комментарий в XML-документе:
            new XComment("Телефонная_книга - это корневой элемент XML-документа:"),
            new XElement("Телефонная_книга", // - имя корневого элемента

              new XComment("Элемент СТРОКА содержит атрибут Контакт и два вложенных элемента"),
              new XElement("СТРОКА", // - имя (Name) элемента
                new XAttribute("Контакт", "Олег"),
                new XElement("Домашний_телефон", "236-23-67"), // - имя элемента и его значение
                new XElement("Мобильный_телефон", "+7(495)625-31-43")),

              new XComment("Атрибут Контакт имеет значение 'Прогноз погоды':"),
              new XElement("СТРОКА",
                new XAttribute("Контакт", "Прогноз погоды"), // - атрибут элемента СТРОКА
                new XElement("Домашний_телефон", "001"),
                new XElement("Мобильный_телефон", "")), // - имя элемента и его зачение (Value)

              new XComment("Поскольку каждый элемент Контакт имеет атрибут и два вложенных=>"),
              new XElement("СТРОКА",
                new XAttribute("Контакт", "Борис Григорьевич"), // - имя атрибута - Контакт
                new XElement("Домашний_телефон", "402-12-45"),
                new XElement("Мобильный_телефон", "+7(495)536-79-94")),

              new XComment("=> элемента, в MS Excel отобразится таблица с тремя колонками"),
              new XElement("СТРОКА",
                new XAttribute("Контакт", "Света"), // - значение атрибута - Света
                new XElement("Домашний_телефон", ""),
                new XElement("Мобильный_телефон", "+7(495)615-24-41")))
                                                    );
                    // Сохранить XML-документ:
                    XMLдокумент.Save(@"C:\имя_файла.XML");
                    Console.WriteLine(XMLдокумент);
                    Console.ReadKey();
                }
            }
        }
```

Чтобы понять текст программы, рассмотрим структуру полученного XML-файла, а для этого откроем этот файл с помощью браузера.

Здесь весь XML-документ вложен в так называемый корневой элемент между начальным тегом <Телефонная_книга> И конечным тегом </Телефонная_книга>. Четыре элемента СТРОКА вложены в корневой элемент. В соответствующей таблице MS Excel элементы СТРОКА будут представлять строку в таблице. В свою очередь элемент СТРОКА содержит в себе атрибут контакт и два вложенных в него элемента, имена (Name) которых — домашний_телефон и мобильный_телефон. Именно поэтому в MS Excel отобразится таблица с тремя колонками (один атрибут и два элемента): "Контакт", «Домашний_телефон» и «Мобильный_телефон».

Элемент может иметь один или несколько атрибутов (а может и не иметь, как, скажем, элемент домашний_телефон), например, первый элемент СТРОКА имеет атрибут с именем (Name) контакт и со значением атрибута (value) — 001.

После запуска данной программы будет выведено на консоль содержимое XML-документа (без XML-объявления), а также будет создан XML-файл. Открыв этот файл с помощью MS Excel, получим таблицу телефонных контактов.

Задания для самостоятельного выполнения:

1. Создать XML файл, содержащий сведения о 5 студентах и их оценках по 5 предметам.
2. Создать XML файл, содержащий сведения о доходах предприятия. Столбцы представлены месяцами, а строки – статьей дохода (5 любых статей)
3. Создать XML файл, используемой для отображения следующей таблицы:

N П/П	ФАМИЛИЯ	ИМЯ	ОТЧЕСТВО	ДАТА РОЖДЕНИЯ	ОБРАЗОВ	ДАТА ЗАЧИСЛЕНИЯ	ДОЛЖНОСТЬ	СПЕЦИАЛЬНОСТЬ
72	Уласевич	Мария	Ивановна	05.01.1955	среднее спец.	01.08.1973	бухгалтер	бухгалтер
73	Юркевич	Любовь	Васильевна	14.07.1957	среднее спец.	25.06.1984	контролер	гидротехник
11	Будыкина	Галина	Константиновна	12.05.1961	среднее спец.	28.04.1990	контролер	техник-электрик
13	Вакарев	Вячеслав	Васильевич	23.03.1941	среднее спец.	16.02.1992	старший контролер	техник радиосвязи
64	Трусов	Игорь	Евгеньевич	12.08.1948	среднее спец.	28.06.1996	контролер	электрик
18	Гринько	Наталья	Никитична	01.01.1960	среднее спец.	24.04.1998	контролер	техник-плановик
5	Бибко	Владимир	Михайлович	07.07.1952	среднее спец.	10.06.1959	техник	техник-строитель
10	Бородько	Татьяна	Николаевна	02.06.1969	среднее спец.	01.06.1986	контролер	инженер-экономист

4 Содержание отчёта:

4.1 Тема работы.
4.2 Цель работы.
4.3 Приборы и оборудование.
4.4 Порядок выполнения работы.
4.5 Выводы.

5 Извлечение значения элемента из XML-документа с использованием LINQ-запросов

1 Цель работы

1.1 Изучить технологию LINQ.
1.2 Научиться извлекать значения элементов из XML-документа с использованием LINQ-запросов.

2 Приборы и оборудование:

2.1 Методические указания.
2.2 ПЭВМ типа IBM PC/XT.
2.3 MS Office.
2.4 Visual C#

3 Порядок выполнения работы:

Имеем XML-данные, в которых содержится таблица с именами и телефонами, причем имена в этой телефонной табличке повторяются, например, строка с именем "Витя" содержит мобильный телефон, а потом по мере знакомства с этим Витей у нас появился уже и его домашний телефон. Задача состоит в том, чтобы в данной таблице телефонов (представленной в виде XML, см. листинг 1) найти все строчки с именем "Витя". Эта маленькая несерьезная, на первый взгляд, задача подразумевает, например, такую уже "серьезную" задачу. Имеем громадную базу данных, которую мы получили на каком-то этапе обработки в виде XML, и нам требуется "отфильтровать" записи в этой базе на предмет содержания в некотором поле определенной строки.

Как видно, в редакторе MS Excel наши XML-данные представлены весьма наглядно. И очень понятно, что мы хотим получить, а именно, все номера телефонов напротив имени "Витя".

3.1 Запустим Visual Studio
3.2 Выбрать проект шаблона **Windows Forms Application**
3.3 Указать имя **Name** — Linq5.
3.4 В конструктор формы, из панели элементов **Toolbox** перетащить текстовое поле **TextBox** для вывода в него найденных строк из таблицы XML.
3.5 В свойствах текстового поля разрешить ввод множества строк (а не одной), для этого свойство Multiline переведем в состояние true. Затем на вкладке программного кода введем текст, представленный в листинге 1.

Листинг 1

```
// Имеем XML-данные, в которых содержатся таблица с именами и телефонами, причем
// имена в этой телефонной табличке повторяются. Задача состоит в том, чтобы в
// данной таблице телефонов (представленной в виде XML) найти все строчки с
```

```
        // именем "Витя" с помощью LINQ-запроса.
        using System.Linq;
        using System.Windows.Forms;
        // Другие директивы using удалены, поскольку они не используются в
данной программе
        namespace Linq5
        {
          public partial class Form1 : Form
          {
            public Form1()
            {
              InitializeComponent();
              this.Text = "LINQ-запрос к XML-данным"; textBox1.Multiline = true;
              // Инициализация XML-строки:
              string СтрокаXML =
             @"<?xml version=""1.0""?>
              <ТаблицаТелефонов>
               <Строка>
                <Имена>Витя</Имена>
                <Номера_телефонов>274 28 44</Номера_телефонов>
               </Строка>
               <Строка>
                <Имена>Андрей</Имена>
                <Номера_телефонов>8-085-456-2378</Номера_телефонов>
               </Строка>
               <Строка>
                <Имена>Карапузова Таня</Имена>
                <Номера_телефонов>445-56-47</Номера_телефонов>
               </Строка>
               <Строка>
                <Имена>Витя</Имена>
                <Номера_телефонов>099 72 161 52</Номера_телефонов>
               </Строка>
               <Строка>
                <Имена>Никипелов</Имена>
                <Номера_телефонов>236-77-76</Номера_телефонов>
               </Строка>
               <Строка>
                <Имена>Зиборов</Имена>
                <Номера_телефонов>254 67 97</Номера_телефонов>
               </Строка>
              </ТаблицаТелефонов>";
              var КорневойЭлемент =
System.Xml.Linq.XElement.Parse(СтрокаXML);
              // Запись строки, содержащей XML в файл:
```

```
            // System.IO.File.WriteAllText("ТаблицаТелефонов.xml",
СтрокаXML);
            // var КорневойЭлемент = System.Xml.Linq.
            //                 XElement.Load("ТаблицаТелефонов.xml");
            var Записи =
                    from x in КорневойЭлемент.Elements("Строка")
                    where (string)x.Element("Имена") == "Витя"
                    select x.Element("Номера_телефонов").Value;
            textBox1.Text = textBox1.Text +
                    @"Строки, содержащие имя ""Витя"":" + "\r\n";
            // Вывод коллекции записей в текстовое поле textBox1:
            foreach (var x in Записи)
               textBox1.Text = textBox1.Text + x + "\r\n";
            // Таких записей в этой коллекции - ровно одна
        }

        private void Form1_Load(object sender, System.EventArgs e)
        {

        }
    }
}
```

Как видно, в начале программы мы инициализируем (т. е. присваиваем начальные значения) XML-строку. Далее извлекаем корневой элемент из XML-документа, он, по сути, отличается от XML-документа отсутствием XML-объявления (в этом можно убедиться в отладчике программы). В комментарии указано, как можно получить корневой элемент в том случае, если он представлен в виде XML-файла во внешней памяти. Затем организуем типовой, стандартный LINQ-запрос. Результат запроса попадает в коллекцию записей, которую выводим в текстовое поле, используя оператор цикла foreach.

Задачи для самостоятельного выполнения:

1. Используя созданный XML-файл в задании №3 из лабораторной работы №4, выбрать информацию о Трусове Игоре
2. Используя созданный XML-файл в задании №1 из лабораторной работы №4, выбрать информацию о каком-либо студенте
3. Используя созданный XML-файл в задании №3 из лабораторной работы №4, выбрать информацию о сотрудниках с должностью контроллер
4. Используя созданный XML-файл в задании №3 из лабораторной работы №4, выбрать информацию о сотрудниках, у которых должностью совпадает со специальностью
5.

4 Содержание отчёта:

4.1 Тема работы.
4.2 Цель работы.
4.3 Приборы и оборудование.
4.4 Порядок выполнения работы.
4.5 Выводы.

6 Создание LINQ – запрос к набору данных DataSet

1 Цель работы

1.1 Изучить технологию LINQ.
1.2 Научиться создавать LINQ-запросы к набору данных DataSet.

2 Приборы и оборудование:

2.1 Методические указания.
2.2 ПЭВМ типа IBM PC/XT.
2.3 MS Office.
2.4 Visual C#

3 Порядок выполнения работы:

Весьма полезной оказывается организация LINQ-запросов к наборам данных DataSet, используемым, к примеру, при работе с базами данных. Объект класса DataSet представляет расположенный в памяти кэш (cache) данных (кэш — это промежуточная память с быстрым доступом, содержащая информацию, которая может быть запрошена с наибольшей вероятностью). Реляционные базы данных работают чаще всего с совокупностью таблиц. Каждая из этих таблиц задается как объект класса DataTable, один такой объект представляет ровно одну таблицу данных. Набор данных DataSet содержит в себе несколько объектов (таблиц) DataTable. Запросы LINQ к таблицам данных, кэшированным в объекте DataSet, упрощают и ускоряют процесс отбора.

Данная задача состоит в том, чтобы создать программу, которая обеспечивает ввод простейшей таблицы, содержащей два поля — название города и численность его населения. Программа способна фильтровать данные в таблице; будем производить отбор городов, численность населения которых превышает миллион жителей.

3.1 Запустить Visual Studio

3.2 Выбрать проект шаблона **Windows Forms Application**

3.3 Указать имя **Name** — LinqГорода.

3.4 В конструкторе формы, из панели элементов **Toolbox** перетащить элемент управления для отображения и редактирования табличных данных **DataGridView,** две командные кнопки **Button** и текстовое поле **TextBox.** Одна кнопка предназначена для команды сохранения данных, другая — для поиска данных в таблице, а текстовое поле — для вывода в него найденных строк из таблицы.

3.4 В свойствах текстового поля разрешить ввод множества строк, для этого свойство Multiline переведем в состояние true. Затем на вкладке программного кода введем текст, представленный в листинге 1.

Листинг 1

```
// В данной программе экранная форма содержит элемент управления для
отображения
```

```
        // и редактирования табличных данных DataGridView, две командные кнопки и текстовое
        // поле. При старте программы, если есть соответствующий файл XML, то программа
        // отображает в DataGridView таблицу городов - название города и численность
        // населения. При щелчке на кнопке "Сохранить" все изменения в таблице записываются
        // в XML-файл. При щелчке на второй кнопке "Найти" выполняется LINQ-запрос к набору
        // данных DataSet на поиск городов-миллионеров в искомой таблице. Результат запроса
        // выводится в текстовое поле.
        using System;
        using System.Collections.Generic;
        using System.ComponentModel;
        using System.Data;
        using System.Drawing;
        using System.Linq;
        using System.Text;
        using System.Windows.Forms;
        // Другие директивы using удалены, поскольку они не используются в данной программе
        namespace LinqГорода
        {
            public partial class Form1 : Form
            {
                DataTable Таблица = new DataTable(); // Создание объекта таблица данных
                DataSet НаборДанных = new DataSet(); // Создание объекта набор данных
                public Form1()
                {
                    InitializeComponent();
                    base.Text = "LINQ-запрос к набору данных DataSet";
                    button1.Text = "Сохранить"; button2.Text = "Найти";
                    textBox1.Multiline = true;
                    if (System.IO.File.Exists("Города.xml") == false)
                    {
                        // Если XML-файла НЕТ:
                        // Заполнение "шапки" таблицы
                        Таблица.Columns.Add("Город");
Таблица.Columns.Add("Население");
                        // Добавить объект Таблица в DataSet
                        НаборДанных.Tables.Add(Таблица);
                        dataGridView1.DataSource = Таблица;
```

```
            }
            else // Если XML-файл ЕСТЬ:
            {
               НаборДанных.ReadXml("Города.xml");
               // Содержимое DataSet в виде строки XML для отладки:
               // string СтрокаXML = НаборДанных.GetXml();
               Таблица = НаборДанных.Tables["Города"];
               dataGridView1.DataMember = "Города";
               dataGridView1.DataSource = НаборДанных;
            }
         }
         private void button1_Click(object sender, EventArgs e)
         {
            // Щелчок мышью на кнопке "Сохранить" - сохранить файл
Города.xml:
            Таблица.TableName = "Города";
            НаборДанных.WriteXml("Города.xml");
         }
         private void button2_Click(object sender, EventArgs e)
         {
            // Щелчок мышью на кнопке "Поиск" - запрос городов-
миллионеров:
            textBox1.Clear(); // - очистка текстового поля
            var ГородаМлн = from Город in Таблица.AsEnumerable()
                        where Convert.ToInt32(Город.
                              Field<String>("Население")) >= 1000000
                        select new
                        {
                           A = Город.Field<string>("Город"),
                           B = Город.Field<string>("Население")
                        };
            textBox1.Text = textBox1.Text + "Города-миллионеры:\r\n";
            // Вывод результата запроса в текстовое поле textBox1:
            foreach (var Город in ГородаМлн)
               textBox1.Text = textBox1.Text + Город.A + " - " + Город.B + "\r\n";
         }
      }
   }
```

В начале программы создаем объекты классов DataSet и DataTable так, чтобы они были видимыми из всех процедур класса Form1. Далее сразу после инициализации компонентов экранной формы (т. е. после выполнения процедуры InitializeComponent) проверяем, существует ли файл Города.xml, куда мы записываем искомую таблицу. Если файл не существует, т. е. пользователь первый раз запустил нашу программу, то мы создаем таблицу, состоящую из двух полей (колонок): "Город" и "Население", добавляем (Add) эту таблицу в набор данных, а

также указываем таблицу в качестве источника данных (DataSource) для сетки данных dataGridView1. Если же файл Города.xml уже создан, то мы читаем его в набор данных DataSet и из него заполняем таблицу данных, а также этот набор данных указываем в качестве источника для сетки данных.

При обработке события "щелчок мышью на кнопке" ***Запись*** программируем сохранение редактируемой таблицы в файле Города.xml. В процедуре обработки события "щелчок на кнопке" ***Найти*** организуем LINQ-запрос к заполненной пользователем таблице DataTable, являющейся представителем DataSet. Условием запроса является отбор таких полей таблицы, где население превышает миллион жителей. Заметим, что если не задавать условие where, то в результате запроса получим все содержимое источника данных, т. е. все строки таблицы городов. Результат запроса выводим в текстовое поле, используя цикл foreach.

Задачи для самостоятельного выполнения:

1. Имеется список сотрудников с окладами. Выбрать сотрудников, у которых оклад меньше 10000 руб.
2. Используя созданный XML-файл в задании №1 из лабораторной работы №4, выбрать студентов с средним баллом меньше 3.
3. Используя созданный XML-файл в задании №3 из лабораторной работы №4, выбрать информацию о сотрудниках (фамилия, имя, отчество), с должностью контроллер.
4. Имеется сведения о стоимости поездки на автобусе из Астрахани в разные города. Пользователь вводит название города. Программа выводит стоимость поездки.

4 Содержание отчёта:

4.1 Тема работы.
4.2 Цель работы.
4.3 Приборы и оборудование.
4.4 Порядок выполнения работы.
4.5 Выводы.

7 Создание активной Web-страницы на Visual C#. Проверка введенных пользователем числовых данных с помощью валидаторов.

1 Цель работы

1.1 Научиться создавать активную Web-страницу на Visual C#.

1.2 Научиться проверять введенные пользователем числовые данные с помощью валидаторов.

2 Приборы и оборудование:

2.1 Методические указания.
2.2 ПЭВМ типа IBM PC/XT.
2.3 MS Office.
2.4 Visual C#

3 Порядок выполнения работы:

Статичные Web-страницы не способны выполнять даже элементарные арифметические действия. Создадим Web-страницу, демонстрирующую способность как раз складывать числа, введенные пользователем. И одноименно подчеркнем единый поход к разработке Windows- и Web-приложений. В обоих случаях имеем те же инструменты: форму, текстовые поля, метки, командные кнопки. Таким образом, на данной разрабатываемой странице будем иметь два текстовых поля, куда пользователь будет вводить числа, командную кнопку и метку, на которую будет выводиться результат сложения двух чисел.

3.1 Для создания Web-приложения открыть Visual Studio

3.2 В окне **New Project** на вкладке **Installed Templates** (Установленные шаблоны) **Visual C# Web** выбрать шаблон **Empty ASP.NET Web Application.**

3.3 В поле **Name** указать имя данного проекта— Summa, при этом снять флажок **Create directory for Solution:** После щелчка по кнопке **ОК** управляющая среда создаст папку Summa, в которой будет находиться проект нового Web-приложения.

Теперь мы можем приступить к проектированию Web-страницы. Поскольку проект пустой (empty), добавьте к текущему проекту Web-форму. Для этого в пункте меню **Project** выберите команду **Add New Item,** в появившемся окне указать шаблон **Web Form** и щелкнуть на кнопке **Add.** При этом в окне **Solution Explorer** будет добавлен значок WebFormi.aspx. Теперь мы можем проектировать внешний вид формы на вкладке **WebFormi.aspx.** Внизу вкладки увидим переключатели **Design. Source** и **Split,** которые позволяют изменить взгляд на форму, соответственно, в виде визуального проекта **(Design),** в виде HTML- и ASP-разметки **(Source),** а также одновременно обе эти возможности **(Split).**

Принцип перетаскивания из панели элементов **Toolbox** нужных для текущего проекта элементов управления точно такой же, как и при проектировании формы Windows. Поэтому перетащите из панели элементов

управления раздела **Standard** необходимые нам два текстовых поля, командную кнопку и метку. Кроме того, в заголовке Web-страницы предусмотрим текст "Введите два числа". Для этого щелкнем правой кнопкой мыши в пределах проектируемой Web-страницы, в контекстном меню выберите пункт **Properties.** В появившемся окне свойств **Properties** среди свойств документа **(DOCUMENT)** найдем свойство **Title,** которое содержит текст заголовка Web-страницы. Этому свойству присвоим значение "Введите два числа".

Уже на этом этапе посмотрим на внешний вид проекта, для этого попытаемся выполнить этот проект— нажмем клавишу <F5>. В результате Internet Explorer откроет файл WebFormi.aspx, сгенерированный управляющей средой, и мы получим Web-страницу.

Понятно, что пока эта страница является *статической.* Закроем Web-браузер вернемся к проекту Web-страницы. По умолчанию проектируемая Web-страница имеет имя WebForml.aspx, но при желании в окне **Sulution Explorer** (Обозреватель решений), щелкнув правой кнопкой мыши на значке **WebForm1.aspx,** это имя можно изменить. Интерактивность, динамичность Web-страницы обеспечивает aspx.cs-файл программной поддержки, содержащий код на Visual C#. Чтобы перейти на вкладку файла программного кода **WebForm1.aspx.cs** следует, например, воспользоваться контекстным меню проекта Web-формы и выбрать команду **View Code.** Вкладка программного кода здесь такая же, как и при проектировании формы Windows. На вкладке файла **WebForm1.aspx.cs** напишем следующий программный код:

```
//Создание простейшей активной Web-страницы на Visual C# 2010. Web-страница демонстрирует способность складывать числа, введенные пользователем
using System;
// Другие директивы using удалены, поскольку они не используются
//в данной программе
namespace Summa
public partial class WebForml : System.Web.UI.Page
protected void Page_Load(object sender, EventArgs e)
{
// Обработка события "загрузка страницы"
Page.Title= "Введите два числа";
Label1.Text = string.Empty;
Button1.Text = "Найти сумму двух чисел";

protected void Buttonl_Click(object sender, EventArgs e)
{
// Обработка события  «щелчок на кнопке»
double Z =Convert.ToDouble(TextBoxl.Text) +
Convert.ToDouble(TextBox2.Text);
Label1.Text = "Сумма = " + Z;
}
}
```

}

Как видно из текста программы, при обработке события загрузки страницы Page_Load очищаем (string.Empty) текстовую метку и задаем текст на командной кнопке. Обратите внимание, что по умолчанию названия элементов управления (например, Button1) система генерирует с прописной буквы. При обработке события "щелчок на кнопке", используя функцию ToDouble класса convert, преобразуем символы, введенные пользователем (клиентом) в переменные типа double, складываем их и выводим сумму в текстовую метку. Готовая Web-форма будет иметь вид, представленный на рис. 1.

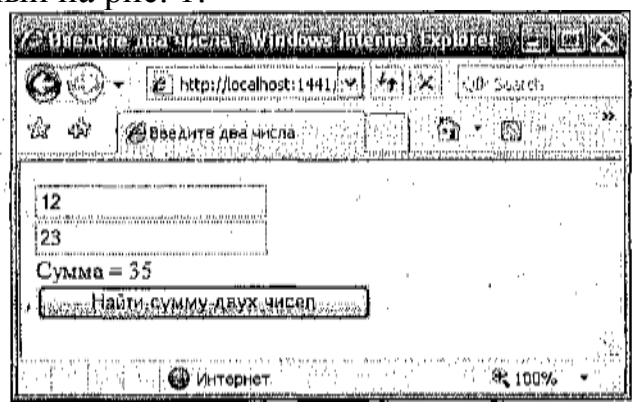

Рисунок 1

В результате работы управляющей среды была скомпилирована библиотека Summa.dll в папке bin. Теперь именно Summa.dll будет обеспечивать интерактивность файлу разметки WebForml.aspx.

Таким образом, созданная Web-страница представляет собой два файла: WebForm1.aspx и WebForm1.aspx.cs. Файл WebForm1.aspx определяет HTML-содержимое сайта, а WebForml.aspx.cs — программный код, который отделен от HTML-содержимого и предназначен для программного формирования данного содержимого или выполнения каких-либо иных действий на сервере. Эта связь определена в файле WebForm1.aspx в директиве Page через свойство CodeBehind= "WebForml.aspx.cs".

Убедиться в работоспособности Web-страницы можно, открыв решение Summa.sln в папке Summa.

Выполнить самостоятельно следующие задания:

1. Определите максимальное из трех введенных чисел
2. Определить корни квадратного уравнения, коэффициенты ввести с клавиатуры
3. Найти среднее арифметическое 3 введенных чисел
4. Ввести пять чисел, определить сколько из них больше нуля
5. Ввести два числа (от и до). Определить количество чисел в этом диапазоне, которые являются четными

Очевидно, что созданная в предыдущем разделе Web-страница имеет много недостатков, в частности если пользователь не заполнил текстовые поля, то получим сообщение "Ошибка сервера...". Между тем следовало бы подсказать пользователю, что надо ввести что-либо в указанные поля. Кроме контроля пустых полей необходимо проверить, является ли введенное значение числом или символом (например, буквой). Для этих целей при разработке Windows-

приложений мы использовали функцию TryParse , однако при разработке Web-приложений имеем среди элементов управления так называемые валидаторы, которые можно использовать в документах ASP.NET для осуществления любой проверки введенных данных. Валидаторы (их шесть) можно увидеть в разделе Validation в панели элементов Toolbox.

В данной задаче мы будем совершенствовать Web-страницу, созданную нами в предыдущем примере. Выполним проверку, ввел ли пользователь хоть что-либо в текстовые поля, а также проверим тип введенных данных и выясним, соответствуют ли они типу данных Double .

3.4 Запустите Visual Studio

3.5 В окне New Project выбрать среди шаблонов Visual C# Web шаблон Empty ASP.NET Web Application

3.6 В поле Name указать имя нового решения — Valid1 и щелкнуть на кнопке OK.

3.7 Добавить к текущему проекту Web-форму. Для этого в пункте меню Project выбрать команду Add New Item, в появившемся окне указать шаблон Web Form и щелкнуть на кнопке Add.

3.8 На вкладке конструктора формы перетащить из раздела Standard панели элементов Toolbox необходимые нам два текстовых поля, командную кнопку и метку.

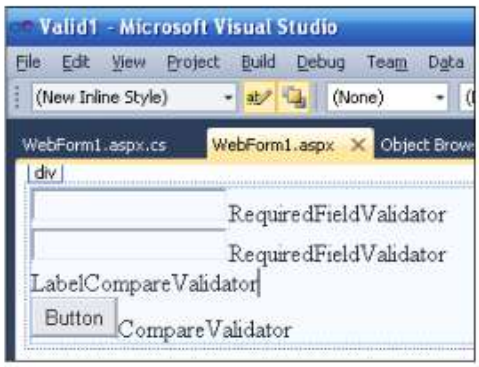

Рис. 13.4. Проектирование Web-формы программы, содержащей валидаторы

3.9 В данной задаче мы должны контролировать два текстовых поля, поэтому перенесите мышью из раздела Validation панели Toolbox соответственно два валидатора RequiredFieldValidator, проверяющих факт заполнения текстовых полей, и два валидатора CompareValidator для проверки, ввел пользователь числа или другие символы. Расположите их в форме, как показано на рис. 13.4.

3.10 На вкладке файла программной поддержки WebForm1.aspx.cs усовершенствовать программный код так, как представлено в листинге 2.

Листинг 2. Использование валидаторов на Web-странице

```
// Проверка введенных пользователем числовых данных с помощью валидаторов.
// Выполним проверку, ввел ли пользователь хоть что-либо в текстовые поля,
// а также проверим тип введенных данных и выясним, соответствуют ли
// они типу данных Double.
using System;
```

```
using System.Web.UI.WebControls;
// Другие директивы using удалены, поскольку они не используются
// в данной программе
namespace Valid1
{
public partial class WebForm1 : System.Web.UI.Page
{
protected void Page_Load(object sender, EventArgs e)
{ // Обработка события "загрузка страницы"
Page.Title = "Введите два числа";
Label1.Text = string.Empty; Label1.Width = 180;
Button1.Text = "Найти сумму двух чисел";
Button1.Width = 180; TextBox1.Focus();
TextBox1.Width = 180; TextBox2.Width = 180;
// Установки валидаторов.
// Контролируем факт заполнения текстовых полей:
RequiredFieldValidator1.ErrorMessage =
"* Следует заполнить это текстовое поле";
RequiredFieldValidator1.ControlToValidate = "TextBox1";
RequiredFieldValidator2.ErrorMessage =
"* Следует заполнить это текстовое поле";
RequiredFieldValidator2.ControlToValidate = "TextBox2";
// Проверяем, соответствуют ли данные, введенные в текстовые поля,
// типу Double:
CompareValidator1.ControlToValidate = "TextBox1";
CompareValidator1.Type = ValidationDataType.Double;
CompareValidator1.Operator = ValidationCompareOperator.DataTypeCheck;
CompareValidator1.ErrorMessage = "* Следует вводить числа";
CompareValidator2.ControlToValidate = "TextBox2";
CompareValidator2.Type = ValidationDataType.Double;
CompareValidator2.Operator = ValidationCompareOperator.DataTypeCheck;
CompareValidator2.ErrorMessage = "* Следует вводить числа";
}
protected void Button1_Click(object sender, EventArgs e)
{ // Обработка события "щелчок на кнопке"
double Z = Convert.ToDouble(TextBox1.Text) +
Convert.ToDouble(TextBox2.Text);
Label1.Text = "Сумма = " + Z;
}
}
}
```

Как видно, при загрузке страницы в свойствах валидаторов RequiredField мы указали имена полей, подлежащих контролю факта их заполнения, а также привели текст сообщения, как реакцию на данную ошибку. В свойствах валидаторов Compare также указали имена полей, подлежащих проверке типа данных DataTypeCheck и текст сообщения.

Пример работы запрограммированных валидаторов приведен на рисунок 2.

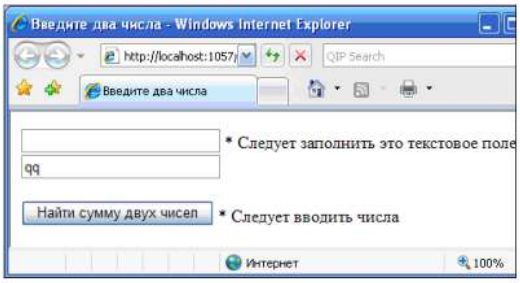

Рисунок 2 - Фрагмент работы Web-страницы с валидаторами

Среди имеющихся на панели Toolbox валидаторов уместно отметить ValidationSummary. Его можно поместить куда-нибудь на Web-страницу, где будут отображаться все сведения об ошибках от разных валидаторов. Причем чтобы сообщения не дублировались, можно указать в свойствах всех валидаторов Visible =false , но только свойство валидатора ValidationSummary.Visible = true .

Очень ценно то, что валидаторы обеспечивают проверку данных не на сервере, а на машине клиента. Это происходит следующим образом: браузер клиента запрашивает на Web-сервере соответствующую страницу; в ответ сервер отсылает на клиентский компьютер HTML-код, который содержит JavaScript с диагностикой, которую мы предусмотрели в cs-файле поддержки. В результате пользователь (клиент) не ощущает задержки во время диагностики данных, поскольку диагностика происходит на клиентском компьютере. Следует отметить то, что если бы мы сами писали валидаторы на языке JavaScript, то потребовалось бы написать гораздо больше строк программного кода.

Убедиться в работоспособности Web-страницы можно, открыв решение Valid1.sln в папке Valid1.

4 Содержание отчёта:

4.1 Тема работы.
4.2 Цель работы.
4.3 Приборы и оборудование.
4.4 Порядок выполнения работы.
4.5 Выводы.

8 Организация раскрывающегося меню гиперссылок. Передача данных между Web -страницами через параметры гиперссылок

1 Цель работы

1.1 Научиться создавать активную Web-страницу на Visual C#.

1.2 Научиться организовывать раскрывающееся меню гиперссылок.

1.3 Научиться передавать данные между Web -страницами через параметры гиперссылок.

2 Приборы и оборудование:

2.1 Методические указания.
2.2 ПЭВМ типа IBM PC/XT.
2.3 MS Office.
2.4 Visual C#

3 Порядок выполнения работы:

Переходы на другие Web-страницы, так называемую навигацию, в статических HTML-страницах обычно организуют с помощью ряда гиперссылок. Например, сайт состоит из десяти Web-страниц. На каждой из этих страниц слева имеем колонку из десяти строчек текстовых гиперссылок для перехода на эти страницы.

В данном примере продемонстрируем, как можно организовать переход на разные страницы сайта с помощью раскрывающегося списка DropDownList.

Для реализации этой задачи:

3.1 Запустите систему Visual Studio

3.2 Новый проект из шаблона Empty ASP.NET Web Application, имя — Menu.

3.3 К текущему проекту добавьте Web-форму.

3.4 В пункте меню Project выберите команду Add New Item и в появившемся окне дважды щелкните на шаблоне Web Form.

3.5 На проектируемую Web-форму перетащите из раздела Standard панели Toolbox элемент управления DropDownList и метку Label.

3.6 Перейдите к вкладке файла программной поддержки WebForm1.aspx.cs и здесь организуйте гиперссылки от каждого пункта меню на некоторые Web-страницы, созданные ранее. Текст программы приведен в листинге 1.

Листинг 1. Организация навигации по Web-страницам

```
// Web-страница демонстрирует, как можно организовать переход на разные
// страницы сайта (гиперссылки) с помощью раскрывающегося списка DropDownList
using System;
using System.Web.UI.WebControls;
// Другие директивы using удалены, поскольку они не используются
```

```
// в данной программе
namespace Menu
{
public partial class WebForm1 : System.Web.UI.Page
{
protected void Page_Load(object sender, EventArgs e)
{
Page.Title = "Переход на другую страницу:";
if (Page.IsPostBack == true) return;
// Можно писать так:
// var it1 = new System.Web.UI.WebControls.ListItem();
// it1.Text = "Сложить два числа";
// it1.Value = "Summa.aspx";
// DropDownList1.Items.Add(it1);
// А можно короче:
// DropDownList1.Items.Add(new ListItem("Имя", "значение"))
DropDownList1.Items.Add(new ListItem(
"Остаться на этой странице", "WebForm1.aspx"));
DropDownList1.Items.Add(new ListItem(
"Проверка достоверности введенных данных",
"Validations.aspx"));
DropDownList1.Items.Add(new ListItem(
"Управляемая таблица", "tab.aspx"));
// Выполнять ли повторную отправку (постбэк), когда
// пользователь сделает выбор в раскрывающемся списке?
DropDownList1.AutoPostBack = true;
// При AutoPostBack = True будет работать
// событие DropDownList1_SelectedIndexChanged.
Label1.Text = "Перейти на другую Web-страницу:";
}
protected void DropDownList1_SelectedIndexChanged(object sender,
EventArgs e)
{ // Выполнить команду перехода на другую страницу:
Response.Redirect(DropDownList1.SelectedValue);
}
//protected void Page_PreInit(object sender, EventArgs e)
//{
// Мы могли бы заполнять пункты раскрывающегося списка DropDownList
// не при загрузке формы, а при ее инициализации (событие
// Page_PreInit), тогда не надо думать про постбэк.
// Чтобы получить обработчик этого события, надо просто его
// написать: protected void Page_PreInit...
//}
}
}
```

Мы заполняли пункты раскрывающегося списка при первой загрузке страницы IsPostBack = false . Это можно было бы делать также при инициализации Web-страницы (событие Page_PreInit).

Причем заполняли список по формату ("Имя" , "Значение"). Здесь "Имя" будет видно в пунктах меню, а "Значение" можно использовать функционально.

Кроме того, для раскрывающегося списка мы указали свойство AutoPostBack =true , что приведет к повторной отправке, когда пользователь сделает свой выбор в раскрывающемся списке, причем в этом случае будет доступно событие SelectedIndexChanged . Именно этим событием мы воспользовались для перенаправления (Redirect) браузера на выбранную пользователем страницу.

Как видно, все выбираемые пользователем Web-страницы являются активными ASPX-страницами. Чтобы добавить их в текущий проект следует:

3.7 Выберите в пункте меню Project команду Add New Item и дважды щелкните на шаблоне Web Form.

3.8 В окне Solution Explorer щелкните правой кнопкой мыши на названии файла и переименуйте новую Web-форму из WebForm2.aspx в Validations.aspx.

3.9 Аналогично добавьте вторую вызываемую Web-форму и назовите ее tab.aspx.

3.10 Перед стартом программы (<F5>) в окне Solution Explorer правой кнопкой мыши щелкните по изображению файла WebForm1.aspx и в контекстном меню выберите команду Set As Start Page, чтобы программа начала работу с этой страницы.

Фрагмент работы данной Web-страницы приведен на рис. 1.

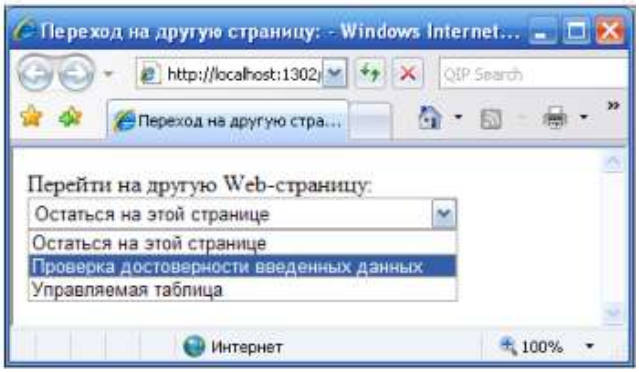

Рисунок 1 - Фрагмент работы Web-страницы, использующей раскрывающийся список

Упомянутая функция Redirect является методом объекта Response . Очень часто пользуются методом Write объекта Response для вывода в Web-форму каких-либо данных, в том числе для тестирования, отладки программы. Вообще говоря, в технологии ASP.NET имеются два очень важных объекта: Response и Request .

Объект Response содержит данные для передачи от сервера клиенту, а объект Request — информацию, полученную от пользователя.

Убедиться в работоспособности Web-страницы можно, открыв решение Menu.sln в папке Menu.

Передача данных между Web-страницами одного сайта может происходить через файлы, через объект Session , а может быть реализована через параметры, указанные в гиперссылке. Эти параметры также называют строкой запроса. Например, гиперссылка

<a href="Target.aspx?n1=77&n2='Зиборов'">

после вопросительного знака содержит два параметра — n1 и n2 . Значения этих параметров будут переданы в Web-документ Target.aspx. Прочитать эти параметры в Target.aspx можно с помощью объекта Request .

3.11Создайте две Web-страницы: Source.aspx и Target.aspx.

3.12 На первой странице Source.aspx с помощью генератора случайных чисел Random выберите одну из пар "имя—фамилия", затем кодируйте их, чтобы они не были видны в

адресной строке.

Щелчок пользователя по гиперссылке вызывает переход на страницу Target.aspx, причем в гиперссылке указаны оба закодированных параметра.

На этой новой странице с помощью объекта Request получаем оба переданных параметра и выводим их в Web-документе.

Для реализации данной задачи:

3.13 Запустите Visual Studio

3.14 В окне New Project выберите шаблон Empty ASP.NET Web Application, в поле Name укажите имя Ssylka.

3.15 К текущему проекту добавьте Web-форму.

3.16 В пункте меню Project выберите команду Add New Item и в появившемся окне дважды щелкните на шаблоне Web Form.

В этом проекте будем программировать две Web-формы: одну назовем Source.aspx, а вторую — Target.aspx. Сначала спроектируем первую форму, для этого в окне Solution Explorer переименуем файл WebForm1.aspx в Source.aspx.

Теперь на вкладку Source.aspx из раздела Standard панели Toolbox перенесем элемент управления HyperLink (Гиперссылка). Содержимое файла программной поддержки Source.aspx.cs приведено в листинге 2.

Листинг 2. Передача данных между Web-страницами через параметры гиперссылки

```
// Передача данных между Web-страницами через параметры гиперссылки. В данном
// примере имеем две Web-страницы: Source.aspx и Target.aspx. На первой
// странице Source.aspx с помощью генератора случайных чисел Random выбираем
// одну из пар "имя-фамилия", затем кодируем их, чтобы они не были видны
// в адресной строке. Щелчок пользователя по гиперссылке вызывает переход
// на страницу Target.aspx, причем в гиперссылке указаны оба закодированных
// параметра
using System;
// Другие директивы using удалены, поскольку они не используются
```

```
// в данной программе
namespace Ssylka
{
public partial class Source : System.Web.UI.Page
{
protected void Page_Load(object sender, EventArgs e)
{
// var Генератор = new Random(DateTime.Now.Millisecond);
// или просто
var Генератор = new Random();
int ЧИСЛО = Генератор.Next(0, 3);
string t1 = null, t2 = null;
switch (ЧИСЛО)
{ // Случайное попадание на одну из меток case:
case 0:
t1 = "Витя"; t2 = "Зиборов"; break;
Глава 13
300
case 1:
t1 = "Света"; t2 = "Ломачинская"; break;
case 2:
t1 = "Андрей"; t2 = "Зиборов-младший"; break;
}
// Закодируем t1 и t2 для того, чтобы они не были видны
// в адресной строке:
t1 = System.Web.HttpUtility.UrlEncode(t1);
t2 = System.Web.HttpUtility.UrlEncode(t2);
HyperLink1.Text = "Щелкните эту ссылку для " +
"передачи данных на страницу Target.aspx";
HyperLink1.NavigateUrl = "Target.aspx?n1=" + t1 + "&n2=" + t2;
}
}
}
```

Как видно из текста программы, при обработке события загрузки страницы Page_Load создаем объект Генератор класса Random , в скобках задаем начало псевдослучайной последовательности, равное, например, как приведено в комментарии, текущей миллисекунде текущей даты. Используя объект Генератор, получаем случайное число в интервале от 0 до 2. Это число, попадая в оператор множественного выбора switch, вызывает присваивание пары "имя—фамилия". Далее функция UrlEncode кодирует t1 и t2 для того, чтобы они не были видны в адресной строке браузера. В конце назначены свойства гиперссылки с указанием параметров n1 и n2 для передачи их в вызываемую Web-страницу Target.aspx. В результате работы Web-формы Source.aspx в браузере мы получим следующее (рис. 2).

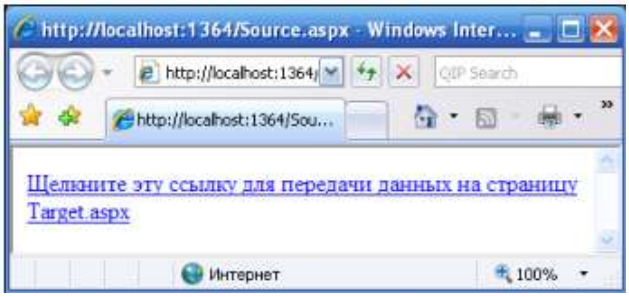

Рисунок 2 - Фрагмент работы Web-страницы с гиперссылкой

Далее, щелкнув на этой ссылке, мы попадаем на вызываемую Web-страницу Target.aspx. Однако в нашем проекте еще нет Web-страницы Target.aspx. Для добавления новой Web-формы в пункте меню Project выберем команду Add New Item и появившемся окне дважды щелкнем на значке шаблона Web Form. В окне Solution Explorer переименуем (щелкнув правой кнопкой мыши) новую Web-форму из WebForm2.aspx в Target.aspx. Добавлять в эту форму из панели Toolbox ничего из элементов управления не станем, поскольку выводить в форму некоторый текст будем, используя метод Write объекта Response . В листинге 3 приведено содержимое файла программной поддержки Target.aspx.cs.

Листинг 3. Получение переданных параметров

```
// Передача данных между Web-страницами через параметры гиперссылки. На этой
// новой странице Target.aspx.cs с помощью объекта Request получаем оба
// переданных параметра и выводим их в Web-документе
using System;
// Другие директивы using удалены, поскольку они не используются
// в данной программе
namespace Ssylka
{
public partial class Target : System.Web.UI.Page
{
protected void Page_Load(object sender, EventArgs e)
{ // Получаем параметры, переданные с Web-страницы, с которой была
// вызвана данная страница
string ИМЯ1, ИМЯ2; // Request - это запрос
ИМЯ1 = Request.QueryString.Get("N1");
ИМЯ2 = Request.QueryString.Get("N2");
string URL = Request.UrlReferrer.AbsoluteUri;
Response.Write(@"<br />Вы попали на данную Web-" +
"страницу со страницы: " + URL + "<br />");
Response.Write(@"<br />Страница " + URL + " передала на " +
"данную страницу имя: " + ИМЯ1 + " и фамилию " + ИМЯ2);
}
}
}
```

В этом программном коде извлекаем переданные параметры n1 и n2 (в данном случае прописные и строчные буквы равнозначны) методом QueryString.Get объекта Request . С помощью объекта Request получаем также

абсолютный URL-адрес вызывающей страницы. Для вывода какого-либо текста на Web-страницу мы могли бы воспользоваться элементом управления Label, однако мы вывели текст "Вы попали..." методом Write объекта Response , чтобы продемонстрировать удобство метода Write , например, для отладки программного кода. Тег
 использован нами для того, чтобы начать предложение с новой строки. Перед стартом программы в окне Solution Explorer правой кнопкой мыши щелкнем по изображению файла Source.aspx и в контекстном меню выберем команду Set As Start Page, чтобы программа начала работу с этой страницы. Результат работы Web-страницы приведен на рис. 3.

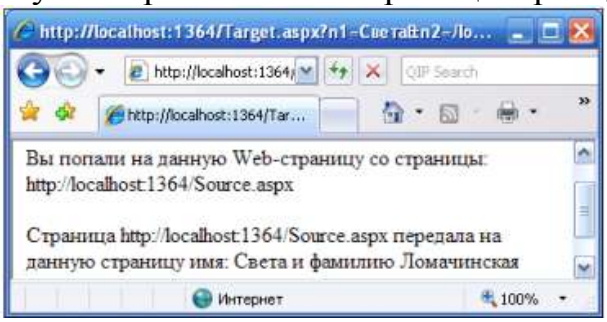

Рисунок 3 - Целевая Web-страница

При использовании строки запроса, т. е. параметров гиперссылки, следует помнить, что некоторые браузеры ограничивают длину URL. Поэтому необходимо следить, чтобы текст в адресной строке не превышал 255 символов.

Убедиться в работоспособности Web-страницы можно, открыв решение Ssylka.sln в папке Ssylka.

4 Содержание отчёта:

4.1 Тема работы.
4.2 Цель работы.
4.3 Приборы и оборудование.
4.4 Порядок выполнения работы.
4.5 Выводы.

9 Отображение времени в Web-форме с использованием технологии AJAX

1 Цель работы

1.1 Научиться создавать активную Web-страницу на Visual C#.
1.2 Научиться отображать время в Web-форме с использованием технологии AJAX.

2 Приборы и оборудование:

2.1 Методические указания.
2.2 ПЭВМ типа IBM PC/XT.
2.3 MS Office.
2.4 Visual C#

3 Порядок выполнения работы:

В Web-форме, технология ASP.NET предполагает операции обратной отправки данных. Когда пользователь щелкает на кнопке или делает выбор в окне списка с включенной функцией AutoPostBack, серверу отсылается соответствующий запрос, после чего с него обратно клиенту передается целая страница. Если такое обращение к серверу будет происходить каждую секунду, то возрастет объем сетевого трафика и вследствие инерции Сети пользователь будет ощущать дискомфорт. Кроме того, если на этой странице предусмотрено заполнение пользователем полей, то серверу будет отсылаться страница с частично заполненными полями.

Решение этой проблемы состоит в использовании технологии AJAX, которая обеспечивает возможность выполнения частичного визуального обновления страницы посредством поддерживаемой ASP.NET AJAX-операции обратной отправки. Продемонстрируем возможность технологии AJAX на примере, когда мы имеем метку **Label** в Web-форме. На эту метку каждую секунду копируем новое время, но обновляем при этом не всю форму, а только метку с помощью технологии AJAX.

3.1 Запустить Visual Studio

3.2 Создать новый проект из шаблона **Empty ASP.NET Web Application,** указать имя **Name** — AjaxTimer.

3.3 К текущему проекту добавить Web-форму. Для этого в пункте меню **Project** выбрать команду **Add New Item** и в появившемся окне дважды щелкнуть по шаблону **Web Form.**

3.4 Из конструктора Web-формы, перетащить в форму из раздела **AJAX Extensions** панели **ToolBox** элементы управления **ScriptManager, Up-datePanel.** Затем на элемент **UpdatePanel** поместим **Timer** и метку **Label.** Теперь на вкладке файла программной поддержки **WebForm1.aspx.cs** напишем программный код, приведенный в листинге 1.

Листинг 1. Отображение времени в Web-форме

```
// Web-страница демонстрирует время на текстовой метке LabeH. На эту метку
// каждую секунду копируем новое время, но обновляем при этом не всю форму,
// а только метку с помощью технологии AJAX using System;
// Другие директивы using удалены, поскольку они не используются
// в данной программе namespace AjaxTimer
{
public partial class WebForm1 : System.Web.UI.Page
{
protected void Page_Load(object sender, EventArgs e)
{
TimerLInterval = 1000;
// 1000 миллисекунд = 1 секунда
LabeH.Text = "Текущее время:  " + DateTime.Now.ToLongTimeString();
}
protected void Timer1_Tick(object sender, EventArgs e)
{
// Обновление содержимого метки каждую секунду:
LabeH.Text = "Текущее время:  " + DateTime.Now.ToLongTimeString();
}
}
}
```

Как видно из программного кода, при загрузке страницы задаем интервал времени, равный одной секунде. При обработке события Tick — события, когда прошел заданный интервал времени, в метку Label1 копируется новое значение времени. При этом благодаря использованию технологии AJAX обновляется не вся Web-форма, а только те элементы, которые расположены на элементе **UpdatePanel.**

Фрагмент работы программы показан на рисунке 1.

Убедиться в работоспособности Web-страницы можно, открыв решение AjaxTimer.sln в папке AjaxTimer.

Рисунок 1 - В форме обновляется только AJAX-элемент **UpdatePanel**

4 Содержание отчёта:

4.1 Тема работы.
4.2 Цель работы.
4.3 Приборы и оборудование.
4.4 Порядок выполнения работы.

4.5 Выводы.

10 Создание простейшей Web-службы

1 Цель работы

1.1 Изучить работу Web-служб на Visual C#.
1.2 Научиться создавать простейшие Web-службы.

2 Приборы и оборудование:

2.1 Методические указания.
2.2 ПЭВМ типа IBM PC/XT.
2.3 MS Office.
2.4 Visual C#

3 Порядок выполнения работы:

Пользователю предлагается ввести два числа, а Web-служба берет на себя функцию сложения этих двух чисел и вывода (возврата) суммы. При этом необходимо произвести диагностику вводимых данных. От такой Web-службы нет особенной пользы, но на этой маленькой задачке нам будет удобно продемонстрировать все возможности. Здесь, так же как и при отладке активных Web-страниц, при отладке данной Web-службы в качестве удаленного серверного компьютера и клиентского компьютера мы будем использовать один ваш локальный компьютер.

3.1 Запустить Visual Studio

3.2 Выбрать шаблон **Empty ASP.NET Web Application,** задать имя, например, WebСлужбаСумма.

3.3 В меню **Project** выбрать команду **Add New Item,** в появившемся окне указать шаблон **Web Service** и щелкнем на кнопке **Add.**

Сразу после этого попадаем на вкладку **WebService1.asmx.cs** готовой Web-службы "Hello World", и уже ее можно тестировать. Содержимое этой вкладки приведено в листинге 1.

Листинг 1. Файл программной поддержки Web-службы "Hello World"

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Web;
using System.Web.Services;
namespace WebСлужбаСумма
{
/// <summary>
/// Summary description for WebService1
/// </summary>
[WebService(Namespace = "http://tempuri.org/")]
```

```
[WebServiceBinding(ConformsTo = WsiProfiles.BasicProfile1_1)]
[System.ComponentModel.ToolboxItem(false)]
// To allow this Web Service to be called from script, using ASP.NET AJAX,
// uncomment the following line.
// [System.Web.Script.Services.ScriptService]
public class WebService1 : System.Web.Services.WebService
{
[WebMethod]
public string HelloWorld()
{
return "Hello World";
}
}
}
```

Однако "HelloWorld — это другая задача, разработчики Visual Studio включили ее для облегчения программистам освоения данной технологии. Нас интересует именно наша задача, поэтому изменим файл программной поддержки, как показано в листинге 2.

Листинг 2. Простейшая Web-служба

```
// На входе данной Web-службы предлагается ввести два числа, а Web-служба
// берет на себя функцию сложения этих двух чисел и вывода (возврата)  суммы.
// При этом Web-служба производит диагностику вводимых данных
using System;
using System.Web.Services;
// Другие директивы using удалены, поскольку они не используются
// в данной программе namespace WebApplication1
{
/// <summary>
/// Summary description for WebService1
/// </summary>
[WebService(Namespace = "http://tempuri.org/")]
[WebServiceBinding(ConformsTo = WsiProfiles.BasicProfile1_1)]
[System.ComponentModel.ToolboxItem(false)]
// To allow this Web Service to be called from script, using ASP.NET AJAX,
// uncomment the following line.
// [System.Web.Script.Services.ScriptService]
public class WebService1 : System.Web.Services.WebService
{
[WebMethod]
public string Сумма(string Число1,  string Число2)
{
// Входные параметры объявляем типа string, чтобы принимать
        // от пользователя любые символы, анализировать их и при
       // "плохом вводе" сообщать по-русски.
Single X, Y;
```

```
        bool Число_ли = Single.TryParse(Число1,
System.Globalization.NumberStyles.Number,
        System.Globalization.NumberFormatInfo.CurrentInfo, out X);
        if (Число_ли == false) return "В первом поле должно быть число";
        Число_ли = Single.TryParse(Число2, System.Globalization.NumberStyles.Number,
System.Globalization.NumberFormatInfo.CurrentInfo, out Y);
        if (Число_ли == false) return "Во втором поле должно быть число";
        Single Z = X + Y;
        return "Сумма = " + Z.ToString();
        }
        }
        }
```

Как видно из текста программы, мы не программировали никакой пользовательский интерфейс. У нас здесь нет ни формы, ни кнопок, ни текстовых полей. Уже отсюда понятно, что Web-служба — это удаленный класс, к которому можно подключиться через Интернет из своего, например, Windows- или Web-приложения. На вход функции Сумма пользователь Web-службы подает два числа, которые принимаются функцией как две строки. Функция Single.TryParse проверяет, числовые ли символы содержатся во введенных строках, и если это так, то происходит преобразование строковых переменных в переменные типа Single.

Теперь можем протестировать данную Web-службу, для этого нажмем клавишу <F5>. При этом Web-браузер откроет ASMX-файл. Здесь мы увидим единственный метод этой Web-службы — Сумма. Щелчок указателем мыши на этой гиперссылке обеспечит возможность тестирования нашей Web-службы (рисунок 1).

Мы можем вводить в данные поля нечисловые символы, числа с десятичной точкой или запятой и тем самым тестировать поведение нашей программы. Заметьте, что мы не программировали эти поля, система сама предоставила нам такую возможность для тестирования программируемой нами Web-службы.

Убедиться в работоспособности данной Web-службы можно, открыв в папке WebСлужбаСумма соответствующий файл решения.

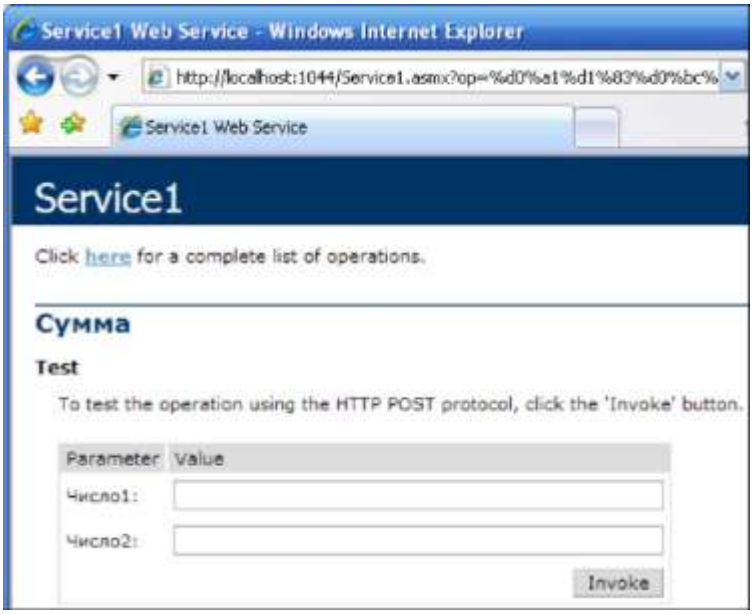

Рисунок 1 - Тестирование созданной Web-службы

4 Содержание отчёта:

4.1 Тема работы.
4.2 Цель работы.
4.3 Приборы и оборудование.
4.4 Порядок выполнения работы.
4.5 Выводы.

11 Создание Windows-приложения — потребителя сервиса Web-службы

1 Цель работы

1.1 Изучить работу Web-служб на Visual C#.

1.2 Научиться создавать Windows-приложения — потребителя сервиса Web-службы.

2 Приборы и оборудование:

2.1 Методические указания.
2.2 ПЭВМ типа IBM PC/XT.
2.3 MS Office.
2.4 Visual C#

3 Порядок выполнения работы:

Теперь напишем Windows-приложение (т. е. создадим exe-файл), который будет обращаться к Web-службе. Такое приложение можно называть *клиентским.*

3.1 Запустить Visual Studio

3.2 Выбрать шаблон **Windows Forms Application C#** (можно даже **Console Application),** указать имя проекта WebКлиентСумма.

3.3 Из панели **Toolbox** перенести в форму командную кнопку **Button.**

Запланируем, что вся работа с удаленной Web-службой будет происходить при обработке события "щелчок на кнопке" **Пуск.**

Чтобы сделать Windows-приложение потребителем сервиса Web-службы, необходимо в его проекте создать Web-ссылку на удаленный класс.

3.4 Выбрать в пункте меню **Project** (или в контекстном меню окна **Solution Explorer)** команду **Add Service Reference,** это приведет к появлению диалогового окна **Add Service Reference.** В этом окне нажать кнопку **Advanced,** а затем в следующем окне кнопку **Add Web Reference.**

Поскольку мы отлаживаем и Web-службу, и его клиентское приложение на одном локальном компьютере, в поле **URL** пишем виртуальный адрес ASMX-файла. Этот адрес мы можем получить, запустив нашу Web-службу из лабораторной работы №10, а затем скопировать в буфер обмена его URL-адрес. Для локального компьютера, на котором автор отлаживал данный пример, URL-адрес был таким: **http://localhost:1611/Service1.asmx.**

3.5 Нажать кнопку **Go** и тем самым получить доступ к кнопке **Add Reference,** после щелчка на которой в окне **Solution Explorer** добавится значок ссылки на класс localhost, обеспечивающий доступ к Web-службе (рисунок 1).

Рисунок 1 - Добавление ссылки на класс localhost

Если посмотреть на этот класс через браузер объектов **View in Object Browser** (используя контекстное меню в окне **Solution Explorer),** то можно увидеть все свойства, методы и события, которые содержатся в этом классе. Теперь, когда в наш проект добавлена ссылка на удаленный класс, мы можем вызывать методы этого класса в нашей программе (листинг 1).

Листинг 1. Windows-приложение, потребляющее сервис Web-службы

```
// Клиентское Windows-приложение, потребляющее сервис Web-службы
// предыдущего примера WebСлужбаСумма
using System;
using System.Windows.Forms;
// Другие директивы using удалены, поскольку они не используются
// в данной программе
namespace WebКлиентСумма
{
public partial class Form1 : Form
{
public Form1()
{
InitializeComponent(); buttonLText = "Пуск";
}

private void button1_Click(object sender, EventArgs e)
{
// Чтобы добавить Web-службу к обычному Windows-приложению:
// Project | Add Service Reference | Advanced | Add Web Reference,
// затем в поле URL пишем виртуальный адрес Web-службы
// http://localhost:1611/Service1.asmx.
// Создаем экземпляр удаленного класса:
var Удаленный = new localhost.WebService1();
string Sum = Удаленный.Сумма("23,5",  "11,4");
MessageBox.Show(Sum);
}
}
}
```

Как видно из программного кода, при обработке события "щелчок мышью" на кнопке **Пуск** создаем экземпляр удаленного класса и обращаемся к его функции (методу) Сумма. При этом при вводе чисел специально одно из них

пишем через десятичную точку, а другое — через запятую для тестирования возможности ввода чисел обоими способами.

Результат работы программы представлен на рисунке 2.

Рисунок 2 - Обращение Windows-приложения к Web-службе

Убедиться в работоспособности программы можно, открыв соответствующий sln-файл решения в папке WebКлиентСумма.

4 Содержание отчёта:

4.1 Тема работы.
4.2 Цель работы.
4.3 Приборы и оборудование.
4.4 Порядок выполнения работы.
4.5 Выводы.

12 Создание простейшего WPF-приложения

1 Цель работы

1.1 Научиться создавать простейшие WPF-приложения.
1.2 Научиться компоновать элементы управления с помощью сетки *Grid*.

2 Приборы и оборудование:

2.1 Методические указания.
2.2 ПЭВМ типа IBM PC/XT.
2.3 MS Office.
2.4 Visual C#

3 Порядок выполнения работы:

Требуется создать интерактивную Web-страницу, содержащую в себе текстовое поле, метку и кнопку. Данное приложение вычисляет значение квадратного корня из числа, введенного пользователем. После щелчка на кнопке приложение производит диагностику введенных символов, и если пользователь действительно ввел число, то на текстовую метку выводим результат извлечения корня. Это совсем маленькая задача, но на ее основе мы сможем рассмотреть все этапы создания WPF-приложения.

3.1 Запустить Visual Studio

3.2 Создать новый проект шаблона **WPF Browser Application C#** (из инсталлированных шаблонов **Windows),** назвать новый проект WpfXbapSqrt. После щелчка на кнопке **OK** попадаем на вкладку **Page1.xaml.**

В WPF-приложении, так же как и в ASPX-приложении, имеем файлы разметки, здесь они написаны на языке XAML. Язык XAML (Extensible Application Markup Language) — это язык разметки для декларативного программирования WPF-приложений. Так же как и в ASP-технолонии кроме xaml-файлов разметки имеем соответствующие xaml.cs-файлы программной поддержки. На вкладке **Page1.xaml** размещают элементы управления, которые перетаскивают из панели элементов **Toolbox** на проектируемую Web-страницу. То есть создание WPF-приложений практически не отличается от создания приложений на основе Windows Forms.

Однако, перед тем как размещать и компоновать необходимые элементы, откроем вкладку **Page1.xaml** в режиме XAML (листинг 1). Обратите внимание на тег <Grid>, его назначение — определять сетку с рядами Row и столбцами Column для размещения в ячейках этой сетки (таблицы) элементов управления. Элемент управления **Grid** иногда называют контейнером макета страницы. Линии сетки (Grid-линии) делают невидимыми showGridLines = false, это позволяет компоновать элементы управления в нужном порядке.

Листинг 1. Проектируемая Web-страница на языке XAML

```
<Page x:Class="WpfXbapSqrt.Page1"
```

xmlns="http://schemas.irdcrosoft.com/winfx/2006/xaml/presentation"
xmlns:x="http://schemas.microsoft.com/winfx/2006/xaml"
xmlns:mc="http://schemas.openxmlformats.org/markup-corapatibility/2006"
xmlns:d="http://schemas.microsoft.com/expression/blend/2008"
mc:Ignorable="d"
d:DesignHeight="300" d:DesignWidth="300" Title="Page1"> <Grid>

</Grid> </Page>

Мы намереваемся управлять этой сеткой из соответствующего файла программной разметки и обращаться к обсуждаемой сетке, как к программному объекту. Однако на данный момент у этой сетки нет имени, поэтому в файле разметки для тега <Grid> зададим имя сетки таким образом: <Grid Name="grid1">. Мы будем использовать это имя позже, а сейчас перенесем на форму из панели элементов **Toolbox** (эту панель можно вызвать, как обычно, комбинацией клавиш <Ctrl>+ +<Alt>+<X>) необходимые нам текстовое поле **TextBox,** метку **Label** и командную кнопку **Button.** Можно было бы, просто последовательно дважды щелкая по упомянутым элементам в панели **Toolbox,** добавлять их на Web-страницу, в этом случае они расположились бы один под другим, как это нам необходимо. Однако хотелось бы иметь гибкую методику размещения элементов управления в любом месте Web-страницы. Для этого как раз мы и упоминали сетку **Grid.** Вообще говоря, для целей размещения элементов на Web-странице имеем ряд компонентов, входящих в состав WPF:

- **Canvas** — дочерние компоненты могут сами управлять своим расположением;
- **DockPanel** — дочерние компоненты выравниваются по краям панели;
- **Grid** — дочерние компоненты располагаются в рядах и колонках;
- **StackPanel** — дочерние компоненты располагаются в одну строку вертикально или горизонтально;
- **VirtualizingStackPanel** — дочерние компоненты выравниваются по одной линии — горизонтальной или вертикальной;
- **WrapPanel** — дочерние компоненты располагаются слева направо и "заворачиваются" на следующую строку, когда заканчивается место на текущей строке.

Прежде чем перейти на вкладку программного кода, зададим, что сетка, в ячейках которой мы собираемся размещать эти три элемента управления, имеет одну колонку и три ряда. Для этого через контекстное меню дизайнера Web-страницы попадаем на вкладку **Properties** (Свойства). И для сетки gridl находим свойство columnDefinitions. Щелкая на многоточии, добавляем **(Add)** одну колонку. Аналогично в строке свойства RowDefinitions добавляем три ряда. Вообще говоря, можно было задать колонку и три ряда в программном коде, но сделать это в свойствах сетки **Grid** совсем просто.

Теперь переходим на вкладку программного кода **Page1.xaml.cs** и вводим текст программы, представленный в листинге 2.

Листинг 2. Содержание файла программной поддержки простейшего WPF-приложения

```
// Данное WPF-приложение вычисляет значение квадратного корня из числа,
// введенного пользователем в текстовое поле. После щелчка на кнопке
// приложение производит диагностику введенных символов, и если пользователь
// действительно ввел число, то в текстовую метку выводим результат
// извлечения корня
using System;
using System.Windows;
using System.Windows.controls;
using System.Windows.Media;
// Другие директивы using удалены, поскольку они не используются
// в данной программе
namespace WpfXbapSqrt
{
/// <summary>
/// Interaction logic for Pagel.xaml
    /// </summary>
public partial class Page1 : Page
{
public Page1()
{
InitializeComponent();
// Строка заголовка Web-страницы:
this.WindowTitle = "Извлекаю корень";
// Компонуем элементы управления с помощью сетки Grid:
gridl.Width = 200;
gridl.Height = 100;
// Сетку grid1 размещаем в левом верхнем углу Web-страницы:
gridl.HorizontalAlignment = System.Windows.HorizontalAlignment.Left;
gridl.VerticalAlignment = System.Windows.VerticalAlignment.Top;
// Удобно для отладки показать линии сетки:
// gridl.ShowGridLines = true;
// Расстояния от краев Web-страницы до сетки
grid1: gridl.Margin = new Thickness(10, 10, 10, 10);
// Цвет объекта
grid1: gridLBackground = Brushes.LightGray;
// Указываем, что текстовое поле расположить в нулевой колонке в
// нулевом ряду:
Grid.SetRow(textBox1, 0); Grid.SetColumn(textBox1, 0);
Grid.SetRow(label1, 1);   Grid.SetColumn(label1, 0);
Grid.SetRow(button1, 2);  Grid.SetColumn(button1, 0);
// Текстовое поле располагаем в левом нижнем углу ячейки, сдвинув
// его на 10 пикселов:
textBox1.HorizontalAlignm.ent =
System.Windows.HorizontalAlignment.Left; textBoxLVerticalAlignment =
System.Windows.VerticalAlignment.Bottom; textBoxLMargin = new Thickness(10, 0, 0, 0);
textBox1.Focus(); labelLHorizontalAlignment = System.Windows.HorizontalAlignment.Left;
```

```
labelLVerticalAlignment = System.Windows.VerticalAlignment.Bottom; labeH.Margin = new Thickness(10, 0, 0, 0); buttonLHorizontalAlignment =
        System.Windows.HorizontalAlignment.Left; buttonLVerticalAlignment = System.Windows.VerticalAlignment.Top; buttonLMargin = new Thickness(10, 0, 0, 0); buttonLContent = "Извлечь корень"; buttonLWidth = 120; labeH.Content = "Введите число в текстовое поле"; labeH.Foreground = Brushes.Blue;
        // - синий цвет текста на метке
        private void buttonl_click(object sender, RoutedEventArgs e)
        {
        Single X;
        // - из этого числа будем извлекать корень
        // Преобразование из строковой переменной в Single:
        bool Число_ли = Single.TryParse(textBox1.Text, System.Globalization.NumberStyles.Number,
        System.Globalization.NumberFormatInfo.currentInfo, out X);
        // Второй параметр - это разрешенный стиль числа (Integer,
        // шестнадцатиричное число, экспоненциальный вид числа и прочее).
        // Третий параметр форматирует значения на основе текущего языка
        // и региональных параметров
        // из Панели управления | Язык и региональные стандарты
        // число допустимого формата.
        // Метод возвращает значение в переменную X.
        if (Число_ли == false)
        {
        // Если пользователь ввел не число:
        labeH.Content = "Следует вводить числа";
        labell.Foreground = Brushes.Red;
        // - красный цвет текста на метке
        return;
        // - выход из процедуры
        }
        Single Y = (Single)Math.Sqrt(X);
        // - извлечение корня
        labell.Foreground = Brushes.Blue;
        // - синий цвет текста на метке
        labeH.Content = string.Format("Корень из {0} равен {1:F5}", X, Y);
        }
        }
        }
```

Как видно из программного кода, задачу компоновки элементов управления в ячейках сетки grid1 решаем сразу после выполнения процедуры Initializecomponent. При обработке события "щелчок на кнопке" button1 проверяем, число ли ввел пользователь в текстовое поле, и выводим результат извлечения корня в метку label1.

Фрагмент работы данного WPF-приложения представлен на рисунке 1.

Теперь очень интересно проверить, как это WPF-приложение будет работать при размещении его на каком-нибудь хостинге, работающем под управлением ОС UNIX (таких хостингов подавляющее большинство, а хостингов из-под Windows значительно меньше). Мы не случайно в самом начале назвали данный проект WpfXbarSqrt, использовав при этом только английские буквы, поскольку имена файлов в ОС UNIX не должны содержать кириллицы. Файлы, которые следует размещать на хостинге, расположены в папке bin\Debug, этих файлов — четыре, они имеют следующие типы: exe, pdb, xbap и manifest. Это исполняемый файл, он содержит скомпилированный код, манифест приложения содержит метаданные, связанные с приложением, и непосредственно xbap-файл, который открывает браузер. В качестве браузера следует использовать Internet Explorer или Mozilla Firefox с соответствующими плагинами.

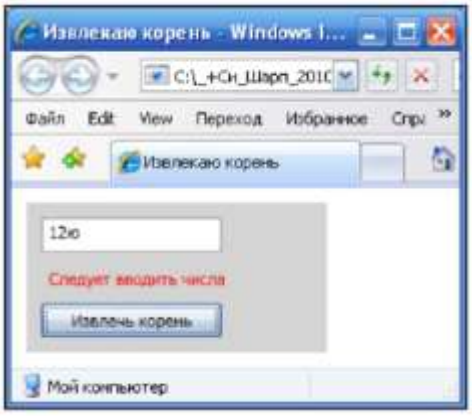

Рисунок 1 - Результат диагностики вводимых данных

Убедиться в работоспособности программы можно, открыв решение в папке WpfXbarSqrt.

4 Содержание отчёта:

4.1 Тема работы.
4.2 Цель работы.
4.3 Приборы и оборудование.
4.4 Порядок выполнения работы.
4.5 Выводы.

13 Использование эффектов анимации в WPF-приложении

1 Цель работы

1.1 Научиться создавать простейшие WPF-приложения.
1.2 Научиться использовать эффекты анимации в WPF-приложении.

2 Приборы и оборудование:

2.1 Методические указания.
2.2 ПЭВМ типа IBM PC/XT.
2.3 MS Office.
2.4 Visual C#

3 Порядок выполнения работы:

Рассмотрим на следующем примере возможность анимации (т. е. способности двигаться) кнопки и изображения при щелчке мышью на них. При щелчке на кнопке она резко расширится, а затем плавно вернется в исходное состояние. Аналогично при щелчке на изображении его размер увеличится с последующим плавным уменьшением до первоначального. Здесь использована наиболее распространенная техника анимации — это анимация интерполяцией, при которой свойство модифицируется плавно от начальной точки до конечной.

3.1 Запустить Visual Studio

3.2 Выбать среди инсталлированных шаблонов Windows шаблон **WPF Browser Application** и назвать его WpfXbapАниме. Чтобы на вкладке **Page1.xaml** появились кнопка и изображение, на панели элементов **Toolbox** дважды щелкните по элементам управления **Button** и **Image.**

3.3 Скопировать в папку проекта изображение *.png, а затем добавить это изображение в проект командой **Project | Add Existing Item.**

3.4 Перейти на вкладку программного кода **Page1.xaml.cs** и ввести текст программы, представленный в листинге 1.

Листинг 1. WPF-приложение демонстрирует одну из возможностей анимации

```
// WPF-приложение выводит на Web-страницу командную кнопку и изображение. При
// щелчке на кнопке и на изображении демонстрируются возможности анимации:
// кнопка расширяется, а затем медленно уменьшается до исходных размеров;
// аналогично поведение изображения
using System;
using System.Windows;
using System.Windows.Controls;
using System.Windows.Input;
using System.Windows.Media.Imaging;
// Другие директивы using удалены, поскольку они не используются
// в данной программе
```

```
namespace WpfXbapАниме
{
/// <summary>
/// Interaction logic for PageLxaml
/// </summary>
public partial class Page1 : Page
{
public Page1()
{
InitializeComponent();
this.WindowTitle = "Анимационный эффект"; buttonLWidth = 100;
buttonLContent = "Привет!";     button1.Focus(); imageLSource = new
BitmapImage(new Uri("poryv.png", UriKind.Relative)); imageLWidth = 100; imageL Height =
100;
}
private void button1_Click(object sender, RoutedEventArgs e)
{
var Аниме = new System.Windows.Media.Animation.DoubleAnimation();
// Изменить размер от 160 до 100 пикселов:
Аниме.From = 160; Аниме-To = 100;
// Продолжительность анимационного эффекта 5 секунд:
Аниме-Duration = TimeSpan.FromSeconds(5);
// Начать анимацию:
button1.BeginAnimation(Button.WidthProperty, Аниме);
private void image1_MouseLeftButtonUp(object sender, MouseButtonEventArgs e)
{
// Щелчок на изображении:
var Аниме = new System.Windows.Media.Animation.DoubleAnimation();
АНМyie.From = 160; Аниме-To = 100; Аниме-Duration = TimeSpan.FromSeconds(5);
image1.BeginAnimation(Image.WidthProperty, Аниме);
image1.BeginAnimation(Image.HeightProperty, Аниме);
}
}
}
```

Сразу после инициализации компонентов страницы InitializeComponent устанавливаем ширину кнопки и размеры изображения равными 100 пекселам. Размеры изображения автор взял из его свойств (контекстное меню изображения **Свойства | Сводка).** При щелчке на кнопке создаваемого WPF-приложения инициализируем новый экземпляр класса DoubleAnimation. Далее указываем, в каких пределах изменять ширину кнопки и продолжительность процесса анимации, и, наконец, командой BeginAnimation начинаем анимационный эффект с командной кнопкой button1. Аналогичные установки и команды выполняем при щелчке мышью на изображении, только анимационным преобразованиям подвергаем одновременно ширину и высоту (Height) изображения.

Убедиться в работоспособности программы можно, открыв решение в папке WpfXbarАниме.

Фрагмент работы программы показан на рисунке 1.

Рисунок 1 - Изображение плавно уменьшается

4 Содержание отчёта:

4.1 Тема работы.
4.2 Цель работы.
4.3 Приборы и оборудование.
4.4 Порядок выполнения работы.
4.5 Выводы.