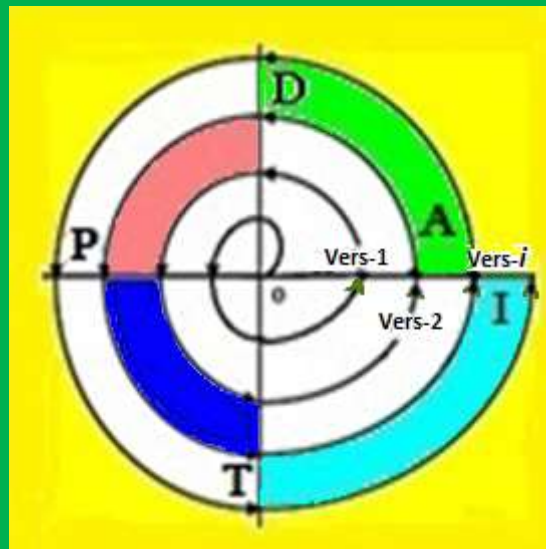


Сургуладзе Гия, Нарешелашвили Гулбаат

# ОСНОВЫ ПРОГРАММНОЙ ИНЖЕНЕРИИ

*(Лабораторный практикум)*





საქართველოს ტექნიკური  
უნივერსიტეტი  
1922 წლიდან

Грузинский Технический Университет  
Факультет Информатики и систем управления

Сургуладзе Гия, Нарешелашвили Гулбаат

# ОСНОВЫ ПРОГРАММНОЙ ИНЖЕНЕРИИ

*(Лабораторный практикум)*



Утверждено:  
редакционной коллегией научного  
центра ИТ-консалтинга ГТУ

Тбилиси - 2023

УДК 004.5

Рассматриваются теоретические и практические задачи визуального программирования и средства их решения на базе новой программной платформы Microsoft Visual Studio.NET 2019/22. Представлены этапы создания и развития программной инженерии и основные технологические решения. Основные визуальные компоненты создания программ и приложений информационных систем управления, типичные трудности процессов разработки и рекомендации по их решению. Учебное пособие предназначено для студентов, изучающих курсы информатики по программной инженерии, информационным системам управления и информационным технологиям, а также для читателей, интересующихся вопросами создания программного обеспечения компьютерных систем.

Рецензенты:

Самхарадзе Р. – Профессор каф. Программной инженерии, Д.т.н.

Туркия Е. – К.т.н. по Информатике. Зав. отделом Национального Банка Грузии

**Редакционная коллегия:**

А. Прангишвили (председатель), З. Азмаипарашвили, М. Ахобадзе, Н. Бераия, З. Босикашвили, Г. Гогичаишвили, Р. Какубава, И. Картвелишвили, Н. Ломинадзе, Т. Ломинадзе, Г. Меладзе, Р. Самхарадзе, Л. Петриашвили, Г. Сургуладзе, Б. Шаншиашвили, О. Шония, А. Цинцадзе, З. Цвераидзе

© ГТУ „ Научный центр IT-Консалтинга”, 2023

**ISBN 978-9941-8-5110-0**

Все права защищены, никакая часть этой книги (будь то текст, фотографии, иллюстрации и т. д.) не может быть использована в любой форме и любыми средствами (электронными или механическими) без письменного разрешения издателя. Нарушение авторских прав преследуется по закону.

## ОГЛАВЛЕНИЕ

<b>Введение</b> .....	<b>3</b>
<b>Глава 1. Консольные и WinForms приложений</b> .....	<b>5</b>
1.1. Лабораторная работа N1. Построение кода C# и отладка в консольном режиме (на платформе VisualStudio .NET 2019/22) .....	5
1.2. Лабораторная работа N2. Работа с формами Windows и визуальные элементы (Button, Label, TextBox) .....	11
1.3. Лабораторная работа N3. Работа с размерами визуальной информации и с координатами их расположения на форме (Size, Location) .....	16
1.4. Лабораторная работа N4. Элемент управления Timer .....	18
1.5. Лабораторная работа N5. Элемент управления – выбор чисел: NumericUpDown .....	19
<b>Глава 2. Контейнерные элементы управления</b> .....	<b>20</b>
2.1. Лабораторная работа N6. Контейнерный элемент - Panel .....	20
2.2. Лабораторная работа N7. Контейнерные и визуальные элементы управления: (CheckBox, RadioButton, GroupBox) .....	22
2.3. Лабораторная работа N8. Контейнерные элементы: GroupBox и TabControl .....	27
<b>Глава 3. Обработка данных строчного типа</b> .....	<b>31</b>
3.1. Лабораторная работа N9. Данные строчного типа string и класс String .....	31
3.2. Лабораторная работа N10. Методы поиска в строке: IndexOf(), LastIndexOf() и IndexOfAny() .....	34
3.3. Лабораторная работа N11. Работа со строками: Insert(), Remove(), Substring() .....	37
<b>Глава 4. Визуальные элементы управления: ListBox, ComboBox</b> .....	<b>41</b>
4.1. Лабораторная работа N12. Визуальные элементы управления: ListBox, CheckedListBox... ..	41
4.2. Лабораторная работа N13. Визуальный элемент управления ComboBox и его свойство DropDownStyle .....	49
4.3. Лабораторная работа N14. Построение информационной системы „Университеты“ .....	52
<b>Глава 5. Компоненты программирования диалоговых процедур</b> .....	<b>59</b>
5.1. Лабораторная работа N15. Стандартные диалоговые средства языка C# .....	59
5.1.1. OpenFileDialog .....	59
5.1.2. SaveFileDialog .....	60
5.1.3. FolderBrowserDialog .....	61
5.1.4. ColorDialog .....	62
5.1.5. FontDialog .....	62
5.2. Лабораторная работа N16. Программные средства построения <i>главного</i> меню .....	64
5.3. Лабораторная работа N17. Программные средства построения <i>графического</i> меню .....	67
5.4. Лабораторная работа N18. Программные средства построения <i>контекстного</i> меню .....	69
<b>Глава 6. Визуальные средства работы с базами данных языка C#</b> .....	<b>73</b>
6.1. Лабораторная работа N19. Элемент управления представлением таблиц DataGridView..	74
6.2. Лабораторная работа N20. Построение C# приложения с базой данных SQL Server драйвером ADO.NET .....	80
6.2.1. Подготовка экспериментальной базы данных пакетом Ms SQL Server .....	80

6.2.2. Подготовка экспериментальной базы данных пакетом Ms SQL Server .....	81
6.2.3. Присоединение к базе данных .....	82
6.2.4. Активизация элемента DataGridView и определение параметров таблицы .....	82
6.2.5. Вопросы программной реализации .....	86
<b>Глава 7. Тестирование и рефакторинг программной апликации .....</b>	<b>89</b>
7.1. Лабораторная работа N21. Модульное тестирование программ .....	89
7.2. Лабораторная работа N22. Рефакторинг: обработка и реорганизация кода .....	96
- Библиографический список .....	100

## Глава 1

### Консольные и WinForms приложений

#### Лабораторная работа N1

##### 1.1. Построение кода C# и отладка в консольном режиме

(на платформе VisualStudio.NET 2019/22)

**Цель работы:** Написание первого C# кода Visual Studio.NET Framework в консольном режиме. Предварительно на диске D:\ создайте фолдер для хранения будущих программных проектов. Активируйте VisualStudio.NET в консольном режиме (Console Application) с помощью C# создайте новый проект, например Lab №1 (имя можете выбрать самостоятельно).

1. Активируйте программу Visual Studio 2019/22, в полученном окне выберите **Create a new Project** (Рис.1.1):

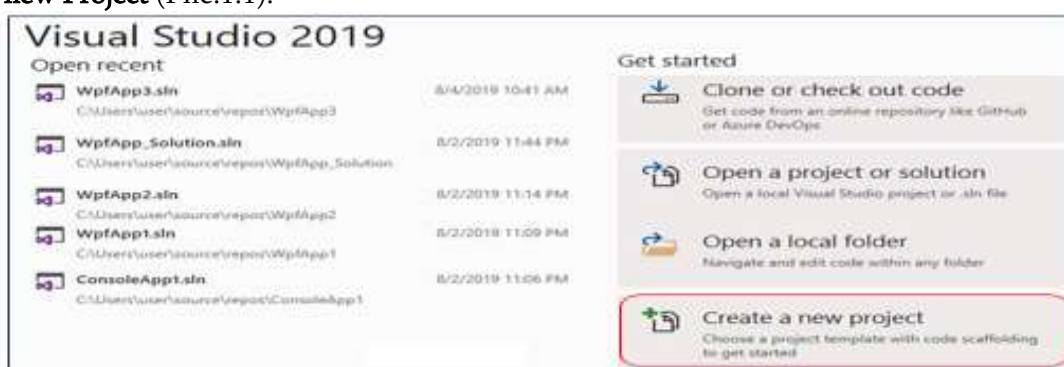


Рис1.1.Создание нового программного проекта.

2. В окне создания нового программного проекта в поле Language в соответствии с примером выберите язык C# (Рис.1.2)

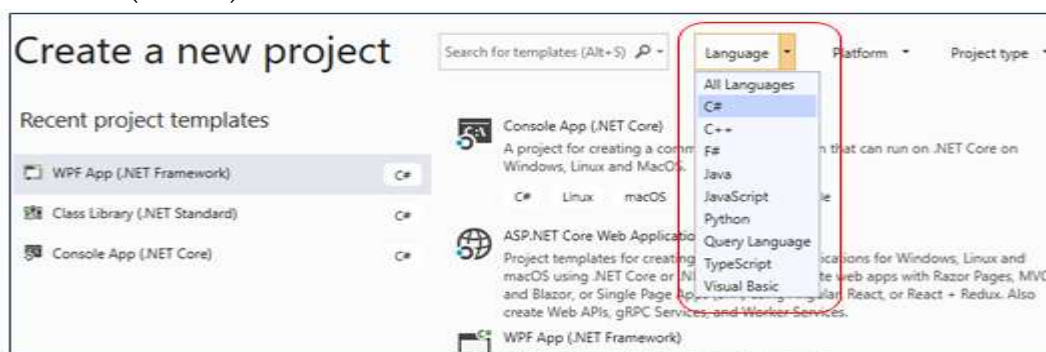


Рис.1.2. Выбор языка программирования для проекта

3. В результате получите выбранную среду C# Console Application (Рис.1.3).

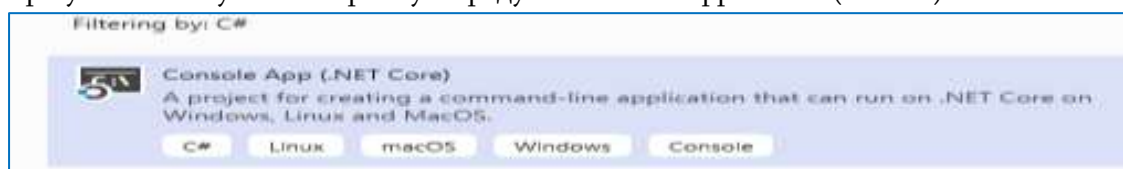


Рис.1.3. Рабочая среда Console App

4. Выбором среды Console App и активизацией Next клавиши, расположенной в нижней правой стороне. Переходите в окно. Где в поле Project Name записывается имя проекта, например, Lab.№1, а в Location выберите имя вашего фолдера, созданного на D диске. (Рис.1.4).

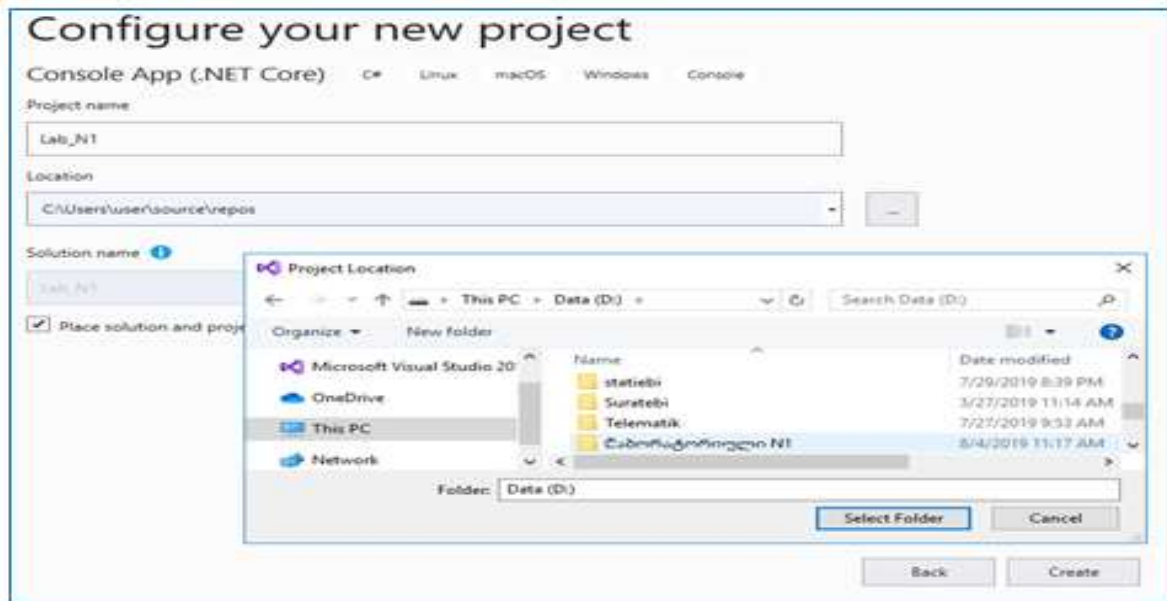


Рис.1.4. Выбор имени проекта и места локации

5. В нижнем правом углу с помощью клавиши Create получите рабочую область, куда заносится код (Рис.1.5).

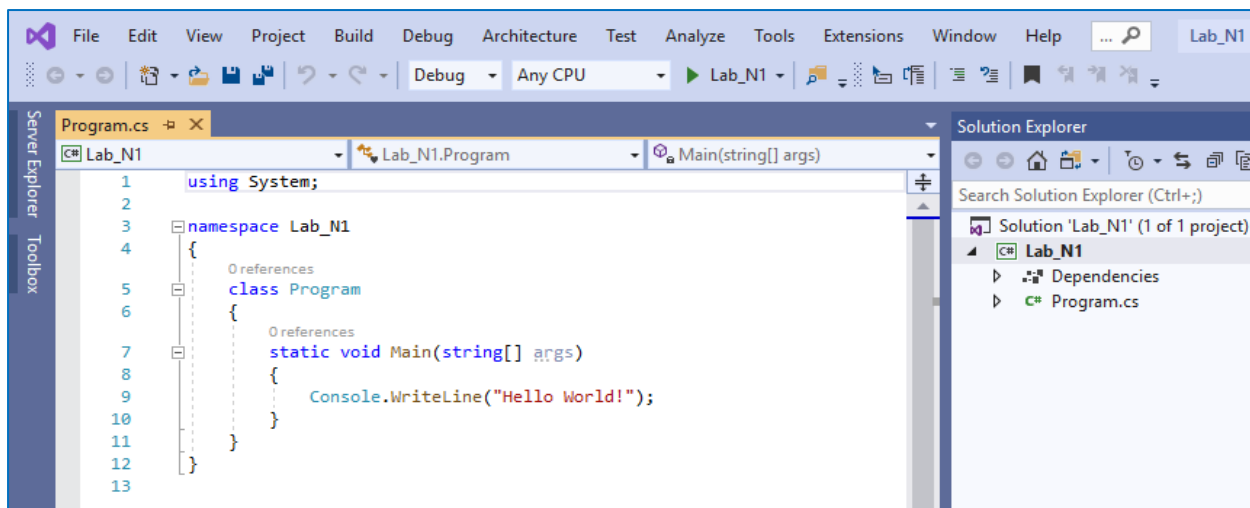


Рис.1.5. Рабочая область для занесения кода

**Задача 1.1:** Пример первой программы. Внесите в код static void Main (...) две строчки (Рис.1.6).

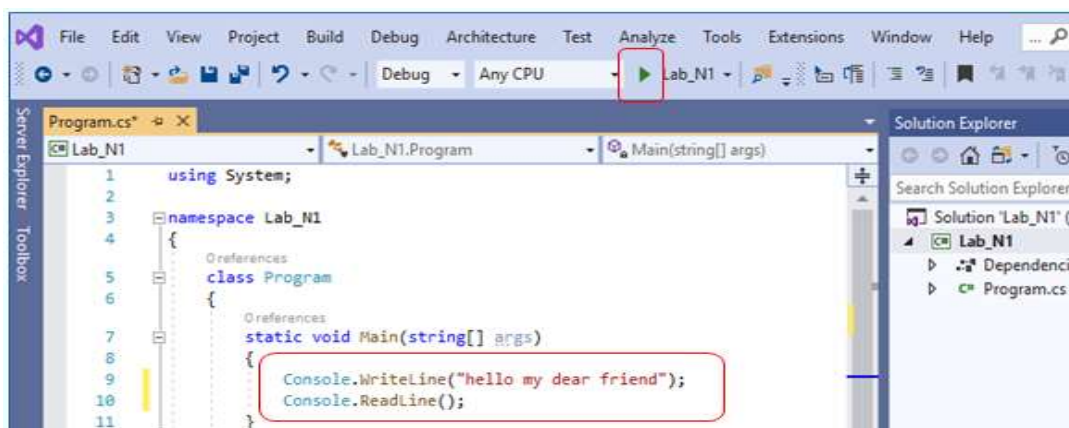


Рис.1.6. Запись методов Write LINE () и Read Line()

Активируйте программу клавишей  ; В результате получите (Рис.1.7).

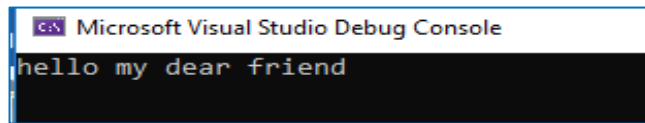


Рис.1.7. Результат

**Задача 1.2.** Пример интерактивного кода ( преобразованием типов данных). С помощью правой клавиши мыши добавим новый проект в Solution “LabN°1” (Рис.1.8).

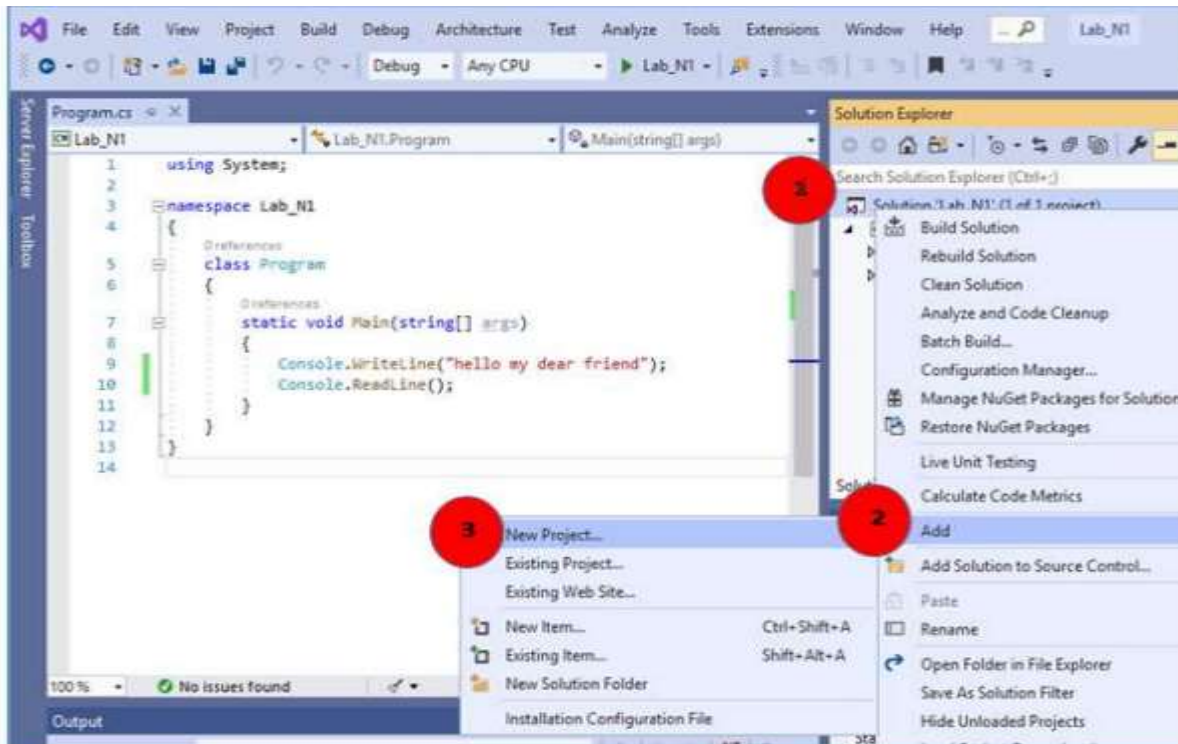


Рис.1.8. Добавление нового проекта

Определим название нового проекта и место его локации с помощью Console App (Рис.1.9).

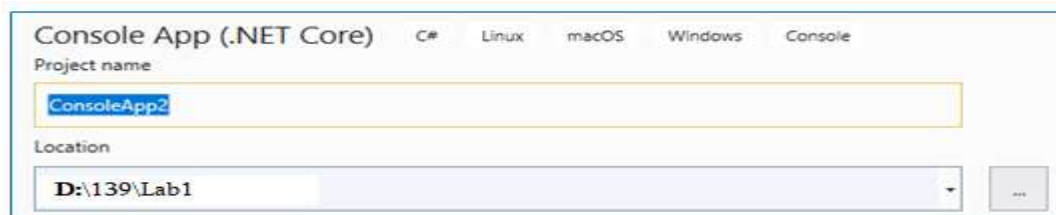


Рис.1.9. Определение наименования проекта и места локации

Внесем C# код (листинг 1.1) с данными: фамилия (Name), возраст (Age) и месячная зарплата (Money). В результате программа выдаст нам консоль эти данные и объем годовой зарплата (Рис.1.10).

**//--- Листинг\_1,1 ----**

```
using System;
namespace ConsoleApp2
{
class Program
{
```



```

static void Main(string[] args)
{
    string Name; int Age=0; decimal Money=0.0m;
    // Input data -----
    Console.WriteLine("\aWhat is your name ? : ");
    Name = Console.ReadLine();
    Console.WriteLine("\aHow old are you ? : ");
    Age = Convert.ToInt16(Console.ReadLine());
    Console.WriteLine("\aYour salary ? : ");
    Money = Convert.ToDecimal(Console.ReadLine());
    // Output results -----
    Console.WriteLine("Hello, {0} !\n", Name);
    Console.WriteLine("Your age={0} years \n", Age);
    Console.WriteLine("Your money={0} dollars \n", Money);
    Console.WriteLine("In Year={0} dollars\n", Money*12);
    Console.ReadLine();
} // в программе использованы методы преобразования
// данных: Convert.ToInt16(), Convert.ToDecimal()
}

```

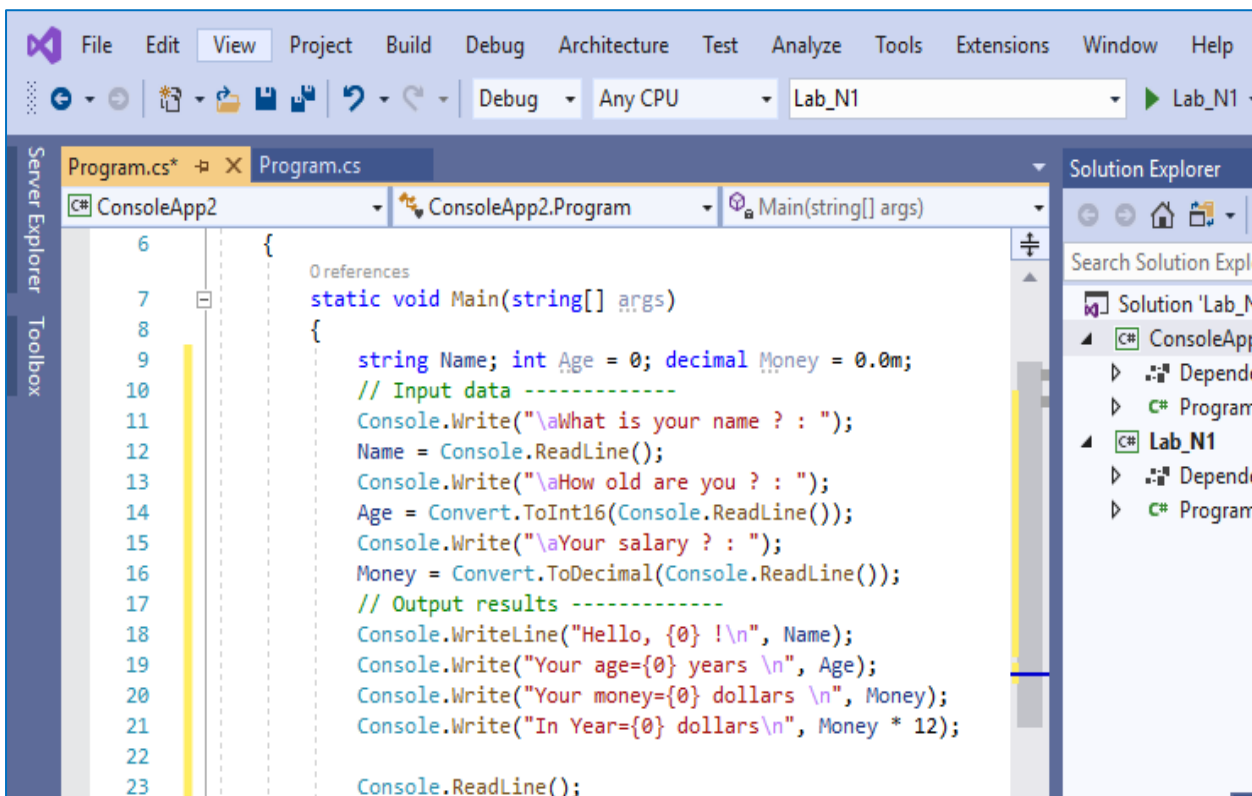


Рис.1.10. Пример кода в рабочей области

Примечание. В результате работы программы получим результат предыдущей задачи (задача 1.1), так как имеем два проекта. Здесь активен Lab\_N1, он обозначен темным цветом. Необходимо активизировать Console App2, правой клавишей мыши из контекстного меню выбираем команду: **Set as Start Up Project** в соответствии с данным внизу образцом (Рис.1.11).

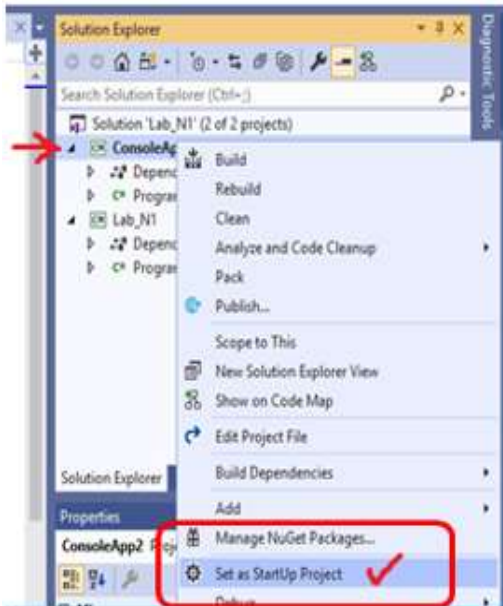


Рис.1.11. Активизация нового проекта (с помощью StartUp)

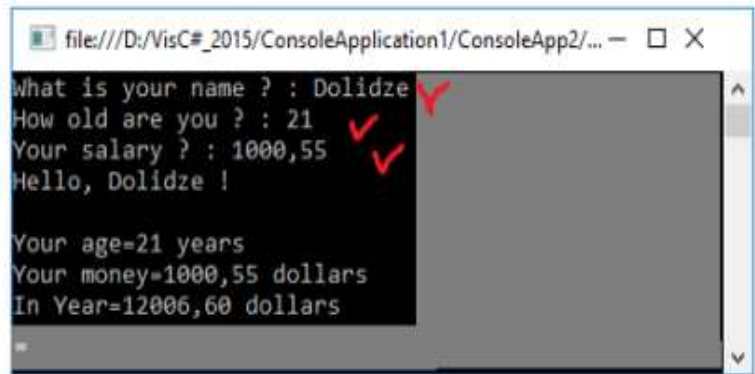


Рис.1.12. Результат работы

После этого активизируем программу, вносим данные в диалоговом режиме и анализируем результат (Рис.1.12).

**Задача 1.3.** Написать интерактивный код на C# для простого калькулятора. В режиме диалога внесите два целых числа и одну арифметическую операцию (+, -, \* или /). Программа выносит результат вычислений. Продолжение программы (циклическое повторение) - командой "Yes", окончание (выход из программы) - командой "No".

Из Solution Explorer добавляем новый проект под именем Console App3 и помещаем в него соответствующий код для решения задачи. Здесь использованы операторы цикла (while(...)) и переключения [switch (op), где op – код арифметической операции]. Текст программы приведен на листинге 1.2.

```
//---Листинг_1.2---
using System;
namespace ConsoleApp3
{
class Program
{
static void Main(string[] args)
{
    int x, y, s;
    char op, yn;
    Console.WriteLine("Calculation - y, End - n: ");
    yn = Convert.ToChar(Console.ReadLine());
    while (yn == 'y' || yn == 'Y')
    {
        Console.WriteLine("Input \n");
        Console.WriteLine("First number: ");
        x = Convert.ToInt32(Console.ReadLine());
        Console.WriteLine("Second number: ");
        y = Convert.ToInt32(Console.ReadLine());
```

```

    Console.WriteLine("Operacia: ");
    op = Convert.ToChar(Console.ReadLine());
    switch (op)
    {
        case '+':
            s = x + y;
            Console.WriteLine("Shedegi: ");
            Console.WriteLine("Sum = " + s);
            break;
        case '-':
            s = x - y;
            Console.WriteLine("Shedegi: ");
            Console.WriteLine("Dif = " + s);
            break;
        case '*':
            s = x * y;
            Console.WriteLine("Shedegi: ");
            Console.WriteLine("Prod = " + s);
            break;
        case '/':
            s = x / y;
            Console.WriteLine("Shedegi: ");
            Console.WriteLine("Div = " + s);
            break;
        case '%':
            s = x % y;
            Console.WriteLine("Shedegi: ");
            Console.WriteLine("Rest = " + s);
            break;
        default:
            Console.WriteLine("operation not Correct !");
            break;
    }
    Console.WriteLine("\nCalculation - y, End - n: ");
    yn = Convert.ToChar(Console.ReadLine());
}
Console.WriteLine("End the process !");
}
}
}

```

Рис.1.13. Результат работы

Фрагмент результатов использования арифметических операций приведен на рис.1.13.

➤ **Самостоятельная работа:**

Создайте в косьном режиме C# код, который в интерактивном режиме даст возможность внести два числа (a,b) и определит их сумму (S), разность (R), произведение (M), частное (D) и модуль (Mod). Результат вынесем на консоль.

## Лабораторная работа N2

### 1.2. Работа с формами Windows и визуальные элементы (Button, Label, TextBox)

**Цель работы:** Ознакомление с визуальными элементами ввода и вывода на экран данных в режиме Windows Forms.

**Задание:** Постройте две формы (Form1, Form2) с помощью следующих элементов: Button, Label, TextBox. В первую форму TextBox введите “слово”. Нажатием клавиши “вторая форма” должна открыться Form2 и в ее TextBox-е появится введенное в первую форму слово. В TextBox-е второй формы исправьте или добавьте новое слово. Нажатием клавиши “Заккрытие” должна закрыться вторая форма и первой форме должна передаться скорректированная строка.

После активизации инструмента Create a new project из рабочей области Visual Studio 2019 пометьте Windows Form App и клавишей Next перейдите в окно, где на созданном вами диске D: в фолдере с именем лаборатория №2 сохраните новый проект – WindowsFormsApp1 (Рис.2.1).

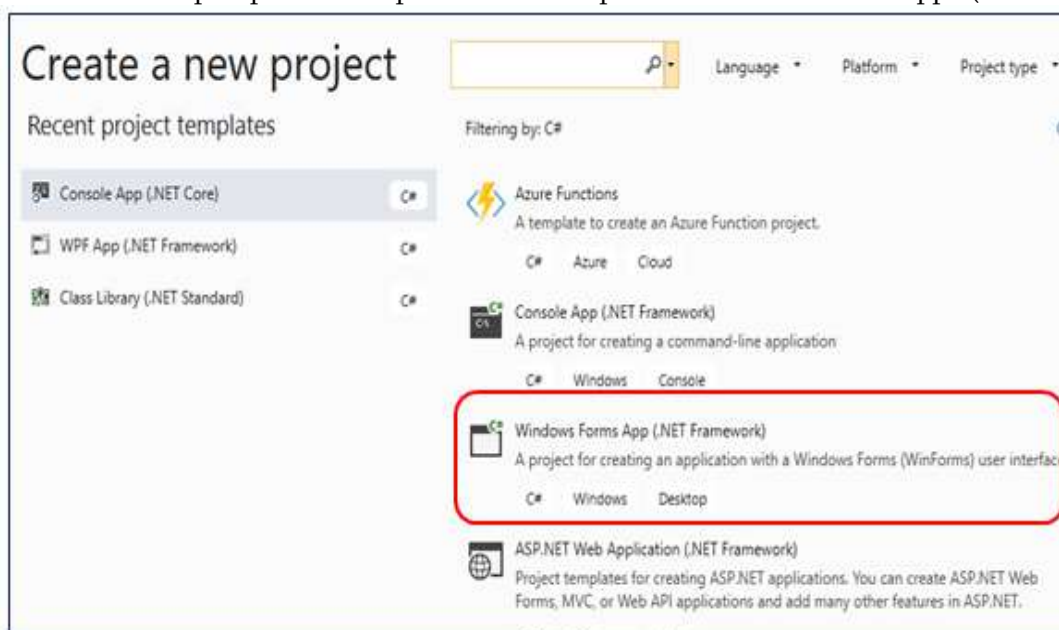


Рис.2.1. Выбор режима Windows Forms App

В результате получим окно, показанное на рис.2.2.

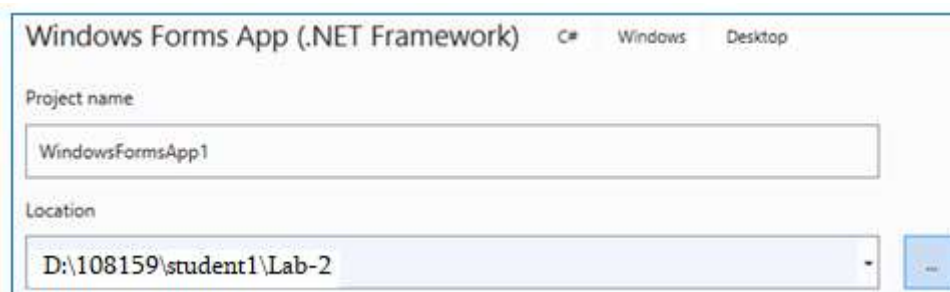


Рис.2.2. Имя проекта и место локации

С помощью клавиши Create, расположенной в нижней правой стороне полученного полученного окна, переходим в область Visual Studio 2019, где есть возможность получить панель инструментов Form1 и ToolBox (Рис.2.3).

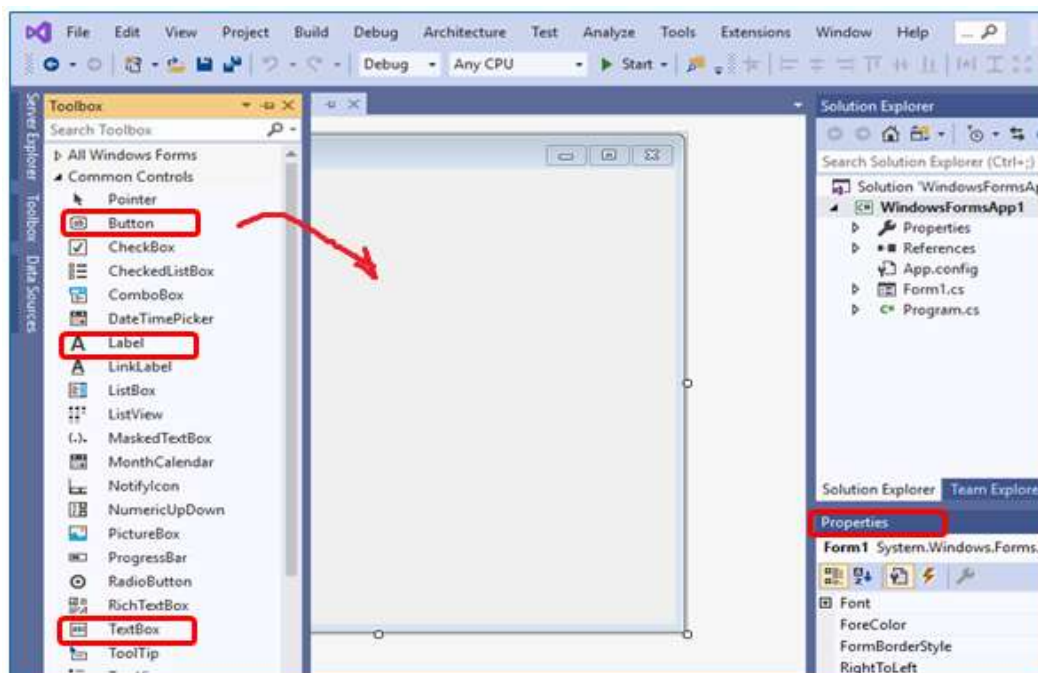


Рис.2.3. Перенос визуальных элементов из инструментальной панели на рабочую форму

Из инструментальной панели, расположенной на левой стороне окна, осуществляется перенос на форму необходимого для программы элемента. В нижней правой части также расположено окно свойств элементов (Properties), которое однозначно описывает свойства формы и ее отмеченного элемента или элементов. После переноса на форму соответствующих инструментов она будет иметь следующую форму (Рис.2.4).

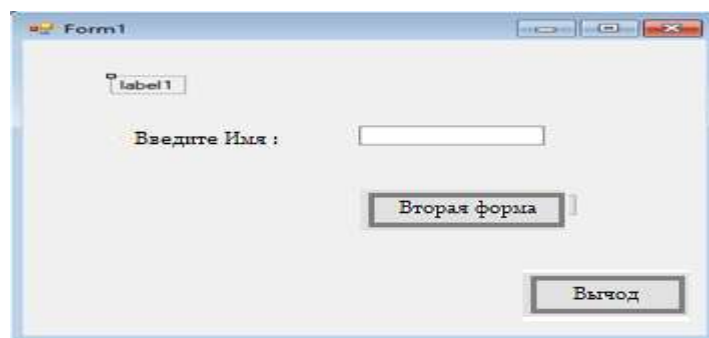


Рис.2.4. Образец построения Form 1

Фрагмент листинга программы первой формы:

```
// WindowsFormsApp1.cs---: 1-формы -----
using System;
using System.Windows.Forms;
namespace WindowsFormsApp1
{
public partial class Form1 : Form
{
public Form1()
{
InitializeComponent();
}
}
```

```
private void button1_Click(object sender, EventArgs)
{
    // вынос введенной строки label1-го -----
    label1.Text = textBox1.Text;
}
}
```

Состав проекта отображается в окне Solution Explorer, где в виде структуры иерархического дерева, расположены его составные элементы: формы, ресурсы, программные коды (Program.cs) и т.д. Здесь возможно добавление новых элементов. Для добавления второй формы необходимо выполнить последовательно следующие команды (Рис.2.5):

WindowsFormsApplication1→Add→WindowsForms

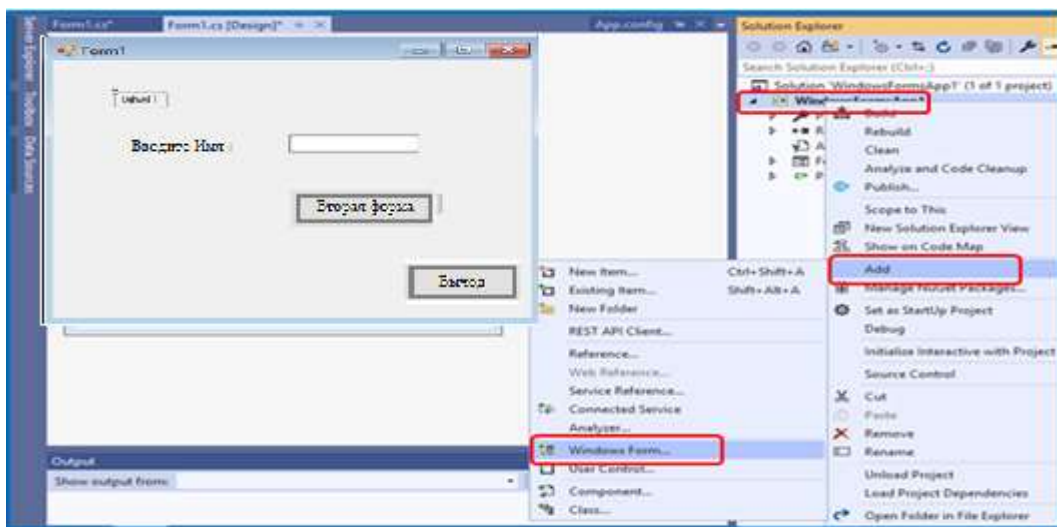


Рис.2.5. Добавление в проект Form2

Получаем окно, где отмечаем Windows Forms и активируем клавишу Add (Рис.2.6).

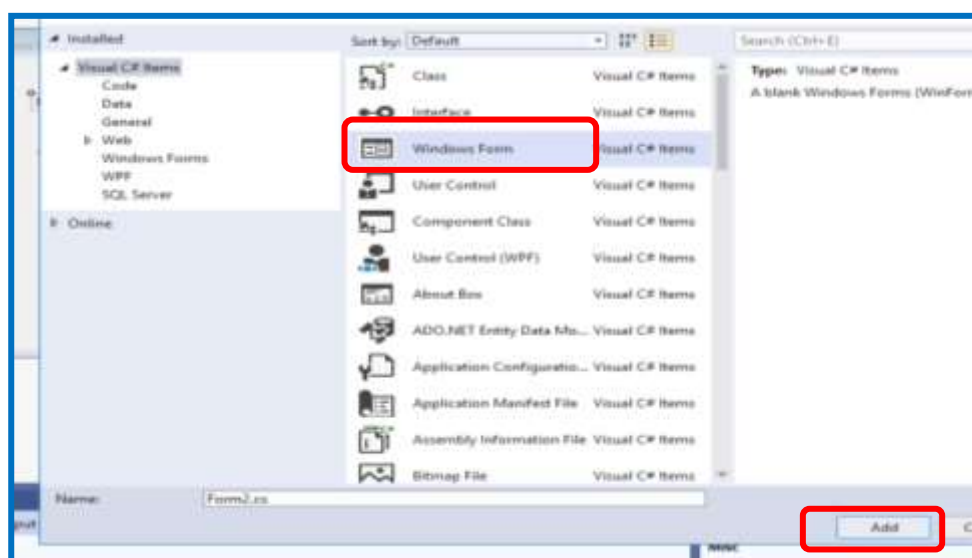


Рис.2.6. Выбор Windows Forms для второй формы

В результате получаем окно для Form2 (Рис.2.7).

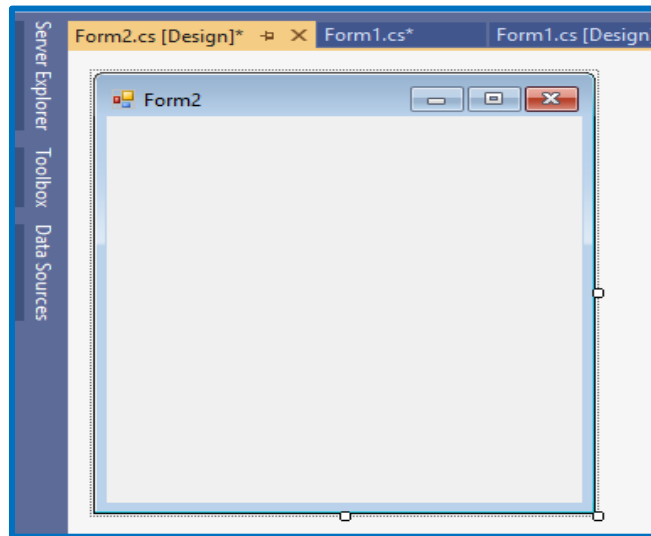


Рис.2.7. Результат с пустой Form2

Как видно из ниже расположенного примера (Рис.2.8), пользователю предоставляется возможность расположить элементы на второй форме и работать одновременно с двумя формами.

В код Form1 необходимо добавить информацию о Form2. В частности, создание ее как нового объекта (f2), ее открытие (Show() методом) и передачу первой форме в textBox-е внесенную строку.

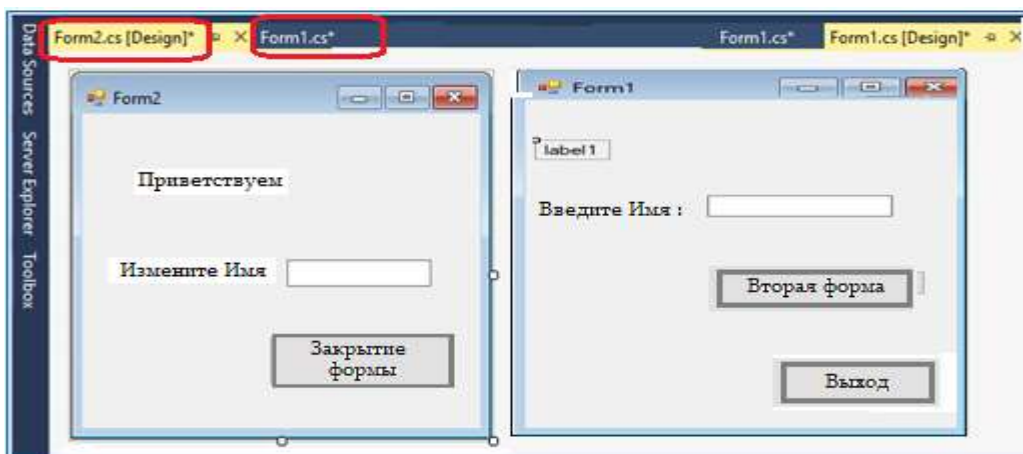


Рис.2.8. Рабочая область с двумя формами

```
// открытиеForm2 из Form1 -----
Form2 f2 = new Form2(this); // !!! this
f2.Show(); // Dialog(); //20
f2.Controls["textBox1"].Text = textBox1.Text;
// вывод сообщения
MessageBox.Show(textBox1.Text+"\",пзакройте это окно !");
}
private void button2_Click(object sender, EventArgs e)
{
    Close();
}
}
}
```

В программный код Form2 необходимо добавить информацию о Form1. В частности, что f1 есть объект главной формы Form1, отобразить из Form1 переданную информацию в своем textBox-е (Form2\_Load).

В textBox-е Form2 после изменения строки клавишей “закрытие формы”, информация должна вернуться textBox-у Form1. Указанный код приведен в листинге:

```
// WindowsFormsApp1.cs---: для 2-ой формы ---
using System;
using System.Windows.Forms;
namespace WindowsFormsApp1
{
    public partial class Form2 : Form
    {
        Form1 f1;
        public Form2(Form1 mainForm)
        {
            f1 = mainForm;
            InitializeComponent();
        }
        private void Form2_Load(object sender, EventArgs e)
        {
            textBox1.Text=f1.Controls["textBox1"].Text;
        }
        private void button1_Click(object sender, EventArgs e)
        {
            f1.Controls["textBox1"].Text = textBox1.Text;
            Close();
        }
    }
}
```

Отладьте и запустите программную аппликацию.

### ➤ Самостоятельная работа:

- Создать новый программный проект с приложением WindowsForms;
- Построить интерфейс на форме с теми элементами, которые описаны в лабораторном задании;
- запишите C# коды для соответствующих клавиш и отладьте их;
- Проведите эксперимент:
  - а) Запустите программу, введите данные согласно инструкции и проанализируйте результаты;
  - б) Добавьте в форму несколько полей textBox-а, введите числовые значения и выполните математические операции. Результаты запишите и проанализируйте;
- Окончательные результаты: программный код проекта и ответы сдайте преподавателю.



## Лабораторная работа N3

### 1.3. Работа с размерами визуальной информации и с координатами их расположения на форме ( Size, Location )

**Цель работы:** ознакомление со средствами управления отображением элементов Windows форме с учетом размеров соответствующих объектов и их местоположения и размещения в окне.

Совершенствование дизайна Windows Forms и эффективное построение управляющих элементов возможно применением различных средств системы, например:

- формирование элементов вертикально или горизонтально на одинаковом (или необходимом) расстоянии. Все элементы могут быть одновременно помечены (Shift/Ctrl) и затем произвести установку их свойств (например, установить во всех текстовых полях грузинского фонта одинаковый размер и цвет);

- копирование элементов: для быстрого построения проекта удобно один раз подготовленные управляющие элементы применять многократно с помощью копирования ( Ctrl+c и Ctrl+v). В данном случае все свойства элементов переносятся в новый, скопированный элемент и нет необходимости заново их устанавливать в Properties;

- изменение свойства элементов в процессе выполнения: в форме у элементов есть свойства Size:Width/Height (Размер: ширина/высота) и Location: X/Y (Расположение: координаты формы от верхнего левого угла). Размеры даются в пикселях. На рис.3.1 представлена форма с пятью клавишами (с режимами работы и редактирования), а также листинг кода для соответствующих клавиш, с помощью которых возможно изменение размера и местоположения бутонов в режиме работы.

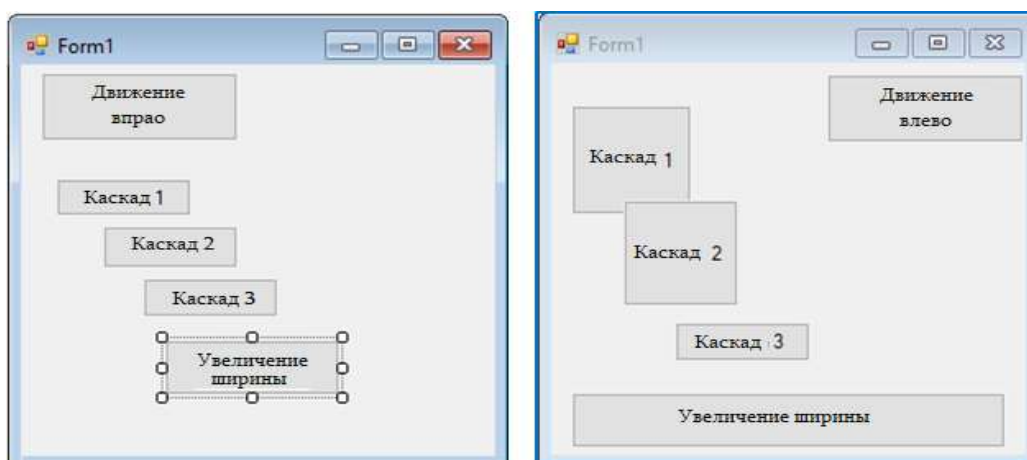


Рис.3.1

- имена элементов: каждый элемент должен иметь свое собственное имя. Рекомендовано перед именем применение трех букв, которые указывают на тип элемента и функциональность. Например, для Label – lbl, для TextBox – txt..., для Button – btn и т.д.

**Задача 1:** постройте форму (Form1) пятью клавишами (button). Для работы каждой клавиши (событие Event) напишите код, который переместит эти клавиши или изменит им размер (см. листинг):

```
// листинг: изменение размеров и расположения элементов формы
using System;
using System.Drawing;
using System.Windows.Forms;
```

```

namespace WindowsMultiButtons
{
    public partial class Form1 : Form
    {
        public Form1()
        {
            InitializeComponent();
        }
        private void Form1_Load(object sender, EventArgs e){

private void button1_Click(object sender, EventArgs e)
    {
        // каскад-1
        button1.Size = new Size(100, 100);
    }
    private void button2_Click(object sender, EventArgs e)
    {
        // переместите вправо на 20 пикселей
        button2.Location = new Point(button2.Location.X + 20, button2.Location.Y);
    }
    private void button3_Click(object sender, EventArgs e)
    {
        // каскад-3
        button3.Location = new Point(120, 220);
    }
    private void button4_Click(object sender, EventArgs

        // увеличение ширины на 20 пикселей одним нажатием на мышь
        button4.Size = new Size(button4.Size.Width + 20, button4.Size.Height);
    }
    private void button5_Click(object sender, EventArgs e)
    {
        // каскад-2
        button5.Size = new Size(100,100);
    }
    }
}

```

➤ **Самостоятельная работа:**

в текстовом элементе формы сцепление многострочного текста и вывод на экран. Символ “ + “ используется для сцепления строк (concatenation). Например, в элементе label1, имя которого lblText (Properties:Name) надо вывести на форму кнопки “каскад-3” строки, которые отображают расположение и размер. На рис.3.2 показана клавиша, с помощью которой получаем в label1 – элементе текстовый результат.

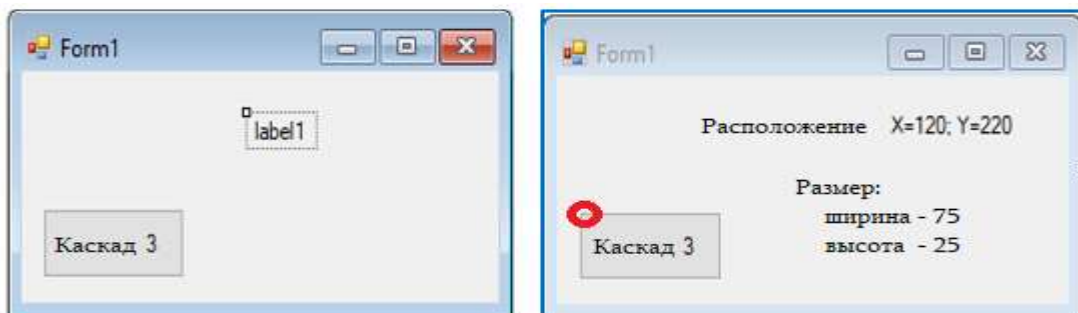


Рис.3.2. Программное определение на форме размеров и координат Местоположения элементов

## Лабораторная работа N4

### 1.4. Элемент управления Timer

**Цель работы:** приведение в действие программных событий, изучение анализа работы их временных показателей с выводом результатов на экран.

Класс **Timer** обеспечивает начало действия события в определенном пользователем интервале. Он используется в приложениях Windows форм. **Имя события Tick** и оно происходит тогда, когда указанный период истек и таймер включен.

На представленном рис.4.1 видна клавиша timer1, нажатием на которую вызывается Form2 и на ней подготовятся две клавиши: привод в действие события Timer-а (Start) и остановки события (Stop). Манипулированием этими клавишами в “Результат“-е выводится полученная строка - конкатенация символа “G” с интервалом 500, что соответствует 0,5 сек.

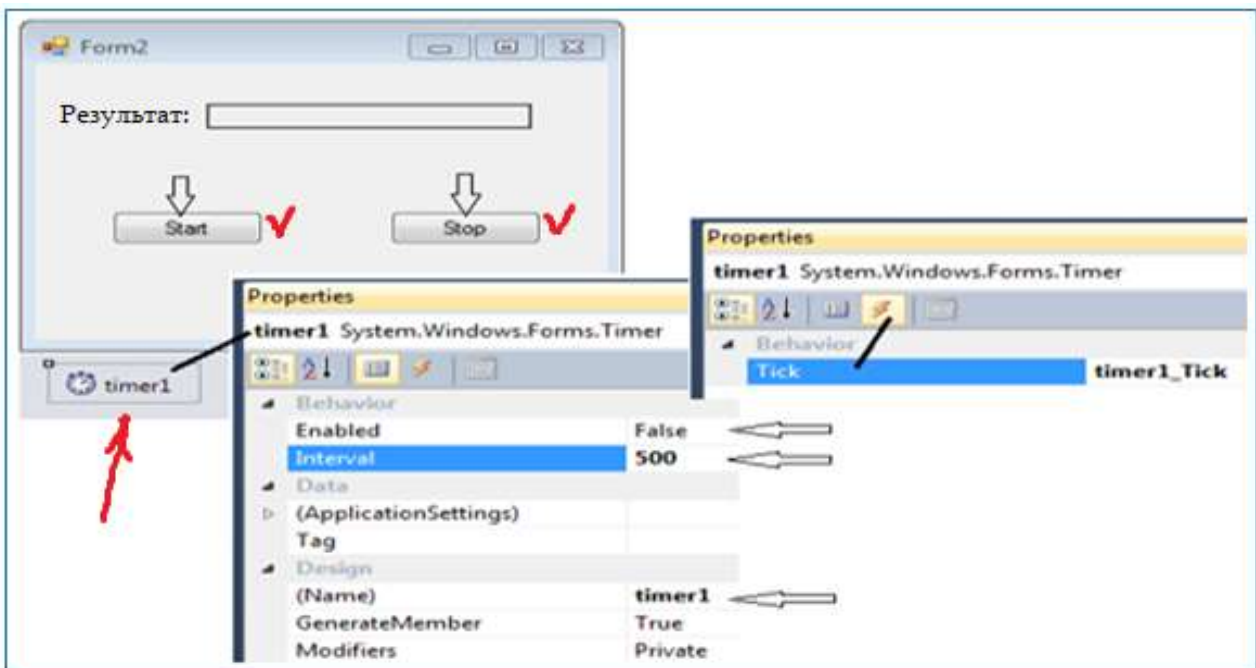


Рис.4.1

На рисунке показана Form2 со своими элементами и свойствами. Timer переносится из панели компонентов (инструментов - Toolbox), он автоматически располагается ниже формы. После его указания в Properties устанавливаются необходимые значения. Текст программы приведен в листинге, где в Form2 расположены Start, Stop и даны их описания. Результат на рис.4.2.

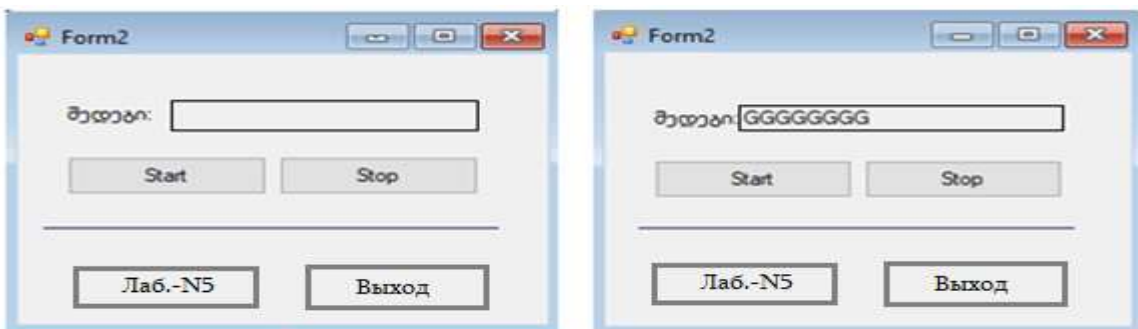


Рис.4.2. Бутон „Лаб.-5“ вызов 5-ой лаборатории (для NumericUpDown)

## Лабораторная работа N5

### 1.5. Элемент управления – выбор чисел: NumericUpDown

**Цель работы:** изучение применения счетного элемента NumericUpDown в программе.

Элемент NumericUpDown дает возможность выбрать необходимое число в режиме счета (маленькие стрелки с правой стороны, с помощью которых осуществляется увеличение или уменьшение исходного числа (Value) с шагом, указанным в Increment-свойстве) или набрать его вручную с клавиатуры. В Properties-е указывается также возможные Maximum и Minimum значения. Результат отображается в label4 (Рис.5.1). Фрагмент кода дан в листинге.

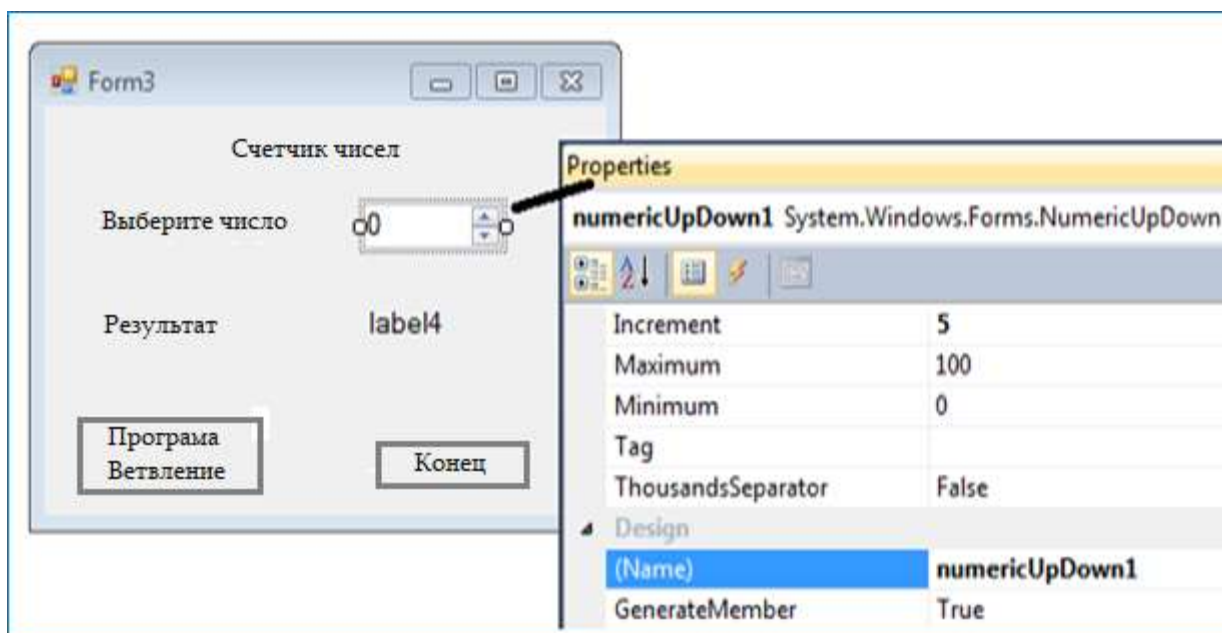


Рис.5.1. Пример для задачи с NumericUpDown

```
// листинг – событие ValueChanged ----
private void numericUpDown1_ValueChanged(object sender, EventArgs e)
{
    label4.Text = numericUpDown1.Value.ToString();
}
```

Окончательный результат дан на Рис.5.2.

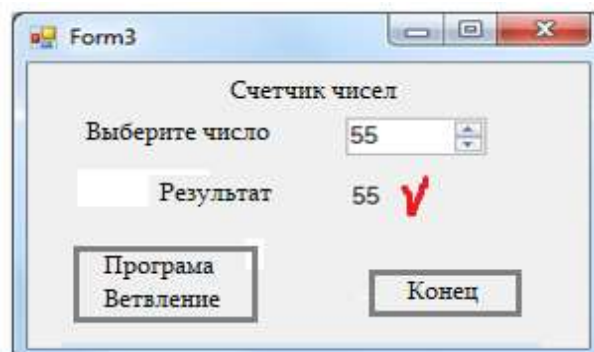


Рис.5.2

#### ➤ Самостоятельная работа:

постройте сравнительно реальную модель калькулятора. Элементами использованы: один textBox элемент (Name = txtDisplay), button-ы; для чисел 0,1,...9; - пять button-ов для операций +, -, \*, /, %; для “плавающая” точка “.” – один button, для фиксации результата “=” - один button и для очистки дисплея “C” (Clear) - один button.

## Глава 2

### Контейнерные элементы управления

Контейнер это такой элемент, который сам содержит в себе другой элемент. Он имеет большое практическое значение и в нескольких параграфах данной части пособия ознакомимся с этими вопросами. На Рис.6.1 показано разнообразие конкретных элементов на панели инструментов Visual Studio.NET.

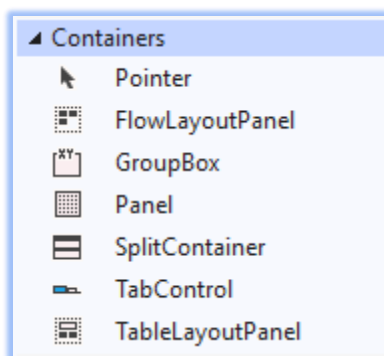


Рис.6.1.Фрагмент Toolbox Containers-a

### Лабораторная работа №6

#### 2.1. Контейнерный элемент - Panel

**Цель работы:** изучение применения контейнера Panel. В контейнер помещаются элементы управления (например, Label, textBox, comboBox и др.). Премещение панели на форме вызывает перемещение вместе с ней компонент, которые помещены в контейнер. Возможно копирование контейнера в другую форму вместе с содержащими в нем компонентами.

**Задача 1:** Рис. 6.2 иллюстрирует пример панели и 4-х клавиш (влево, вправо, вниз, вверх). Параметры (свойства) панели в Properties-е следующие: BackColor=ControlDark // цвет фона панели; Location=120,80 // позиция на форме X=120, Y=80; Size=125,125 // размер панели в пикселях: ширина, высота.

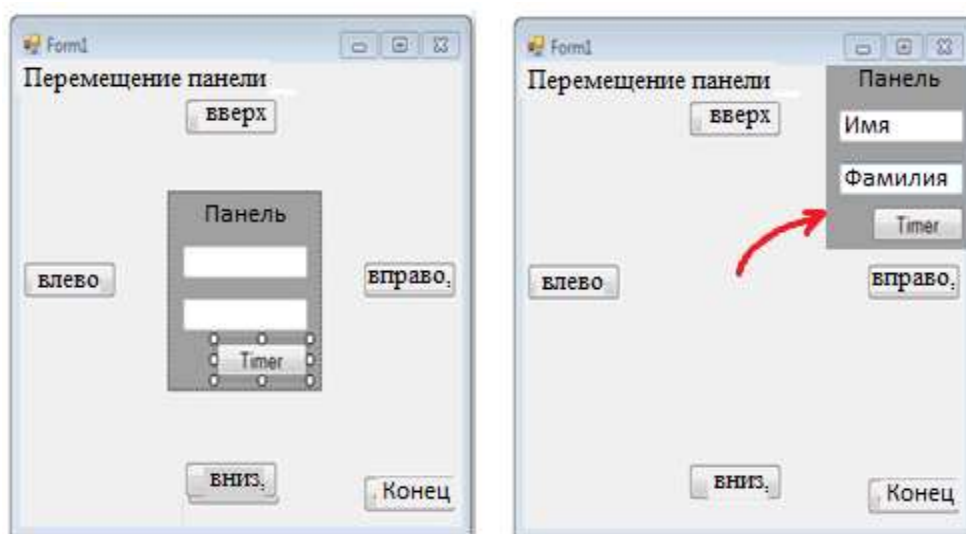


Рис.6.2.

Ниже в листинге приведен код программы, с помощью которой происходит перемещение панели на форме с применением клавиш. На рисунке в верхнем правом углу формы показано конечное положение панели и содержащихся в ней элементов.

**// --- Листинг: Перемещение Panel-ли на форме- ----**

```
using System;
using System.Drawing;
using System.Windows.Forms;
namespace WinFormPanel
{
    public partial class Form1 : Form
    {
        public Form1()
        {
            InitializeComponent();
        }
        private void button1_Click(object sender, EventArgs e) // Top
        {
            panel1.Location = new Point(panel1.Location.X, panel1.Location.Y - 10);
        }

        private void button2_Click(object sender, EventArgs e) // Bottom
        {
            panel1.Location = new Point(panel1.Location.X, panel1.Location.Y + 10);
        }

        private void button7_Click(object sender, EventArgs e) // Left
        {
            panel1.Location = new Point(panel1.Location.X-10, panel1.Location.Y);
        }

        private void button4_Click(object sender, EventArgs e) // Right
        {
            panel1.Location = new Point(panel1.Location.X + 10, panel1.Location.Y);
        }

        private void button5_Click(object sender, EventArgs e)
        {
            Close();
        }
    }
}
```

В тексте программы конструктор `Point()` выполняет инициализацию экземпляра класса с новыми X, Y коэффициентами, после которой объект `panel1` перемещается на форме на 10 пикселей в указанном направлении.

### ➤ Самостоятельная работа:

постройте панель в центре формы, поместите на ней другие элементы, например `textBox` и `label` и переместите в четырех направлениях. При подходе панели к краям окна она должна остановиться (так чтобы не покинуть форму).

## Лабораторная работа N7

### 2.2. Контейнерные и визуальные элементы управления: (CheckBox, RadioButton, GroupBox)

**Цель работы:** изучение свойств и способов применения контейнерных элементов CheckBox, RadioButton и GroupBox.

/  - CheckBox – контрольный ящик или поле, которое выключено (свободно) или отмечено (включено). В случае нескольких CheckBox-ов могут быть включены 0,1 или все.

/  - RadioButton –переключательная клавиша или поле,при включении которой в нее помещается круглый маркер. В случае нескольких RadioButton-ов активен только один, кото-рый отмечен (активизировать два или больше нельзя).

GroupBox является также групповым контрольным ящиком Container класса, ограничен рамкой, в которую можно поместить несколько CheckBox-ов, RadioButton-ов или других визу-альных элементов. Он объединяет однородные данныеб чья текстовая идентификация осуще- ствляется в верхнем левом углу рамки.

**Задача 1:** Необходимо построить виндовс форму, у которой вид, показанный на Рис.7.1. Здесь применен вертикальный Splitcontairn с двумя панелями.

Рис.7.1..Исходная форма

На левой стороне помещена панель с 4-мя CheckBox-ми (вид занятия), две клавиши (“часы”– считает суммарное время заня-тий; “экзамен”- вызывает форму Form2 для новых подзадач).

В верхней части левой панели виден CheckBox “имеет”, который определяет статус студента, т.е.если этот CheckBox выключен это значит, что субъект не является студентом и для него панель управления “вид занятия” выключена (Рис.7.2):

Если CheckBox включен, тогда субъект является студентом и есть возможность на панели “вид занятия” с помощью мыши включить 1,2 или все CheckBox-ы. В label19 часовыми клави-шами возможно показать суммарное число часов этого конкретного предмета в семестре. Если выключить какой-либо CheckBox, тогда суммарное число часов соответственно уменьшится.

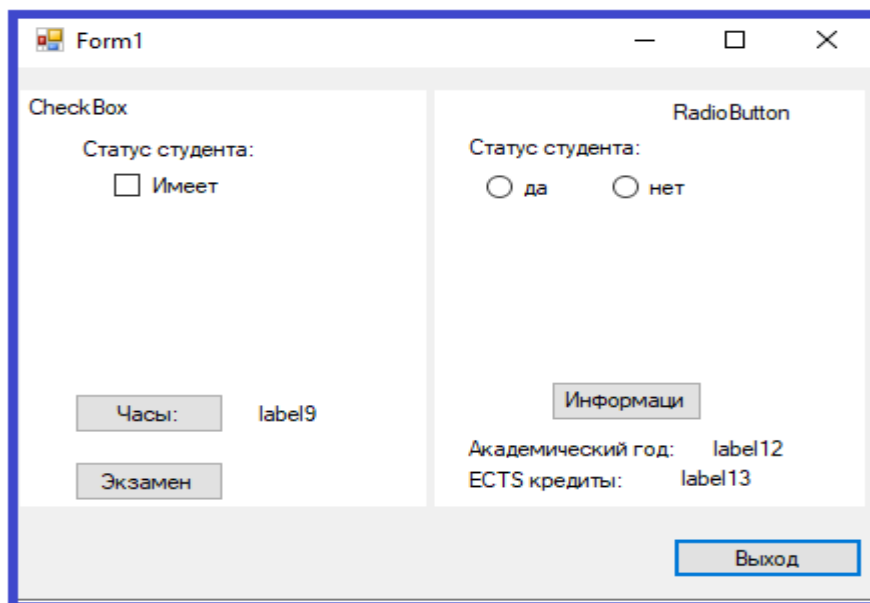


Рис.7.2. Панели выключены

На правой панели показан пример переключателя RadioButton-а. Если в бутоне “да” статуса студента нет маркера, тогда панель “ступень образования” с тремя RadioButton-ами выключена (Рис.7.2). Если есть маркер, тогда эта панель видна и можно выбрать один из буюнов.

Затем приведением в действие клавиши “информация” в полях label2 и label3 появятся значения соответствующих данных: количество учебных годов и общее количество ECTS – кредитов. Переключением RadioButton-ов возможно получение другой информации.

На левой панели включим checkbox статуса. На панели “Вид занятия” включим checkbox – ы “лекция”, “практические” и “лаборатория”. Следующей клавишей “часы” в label19 появится число (Рис.7.3).

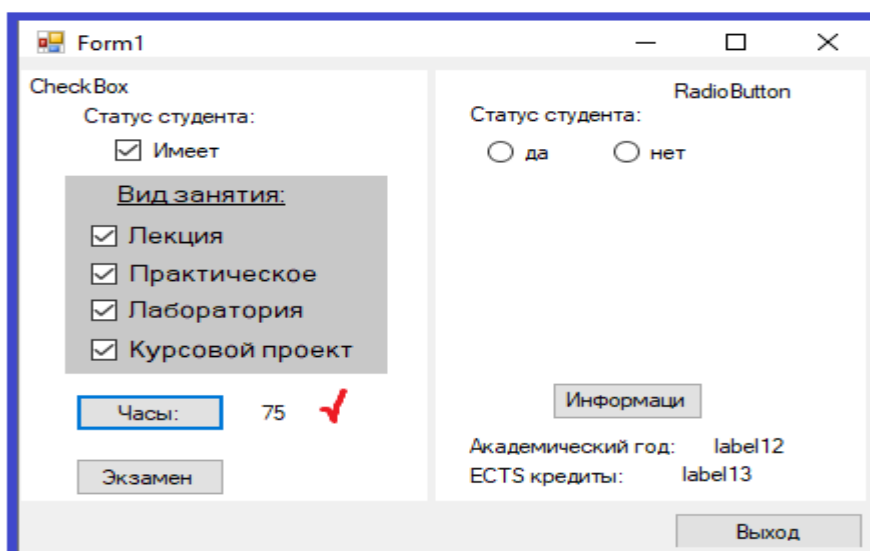


Рис.7.3. Результат: общее количество часов

На рис. 7.4 показан вариант расположения checkbox-ов RowLayoutPanel и TableLayoutPanel с помощью контейнерных элементов.



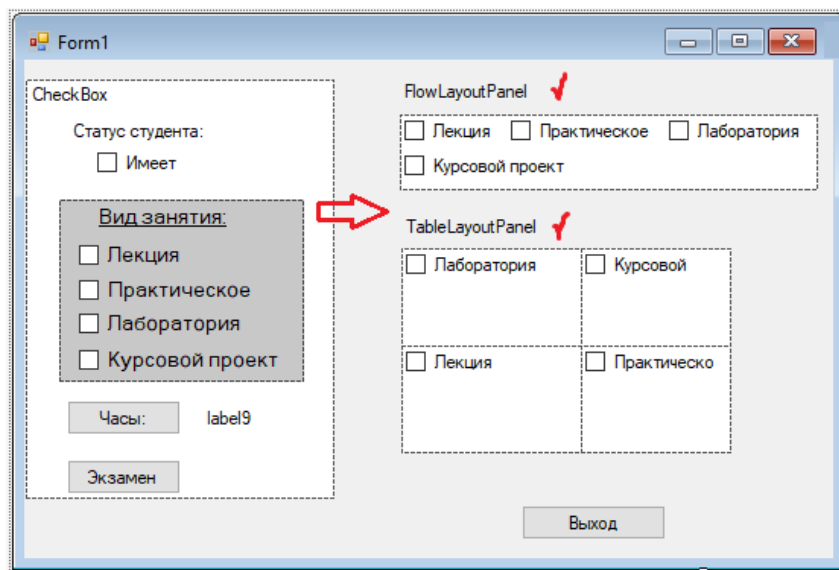


Рис.7.4. Вариант расположения checkbox-ов

Теперь эти же процедуры выполним на примере radiobutton-а в правой части панели. Здесь (Рис.7.3) поместим маркер в “Да” и в появившейся панели “степень образования” выберем какой-либо бутон. Затем активизацией клавиши “информация” получим результат, показанный на рис.7.5.

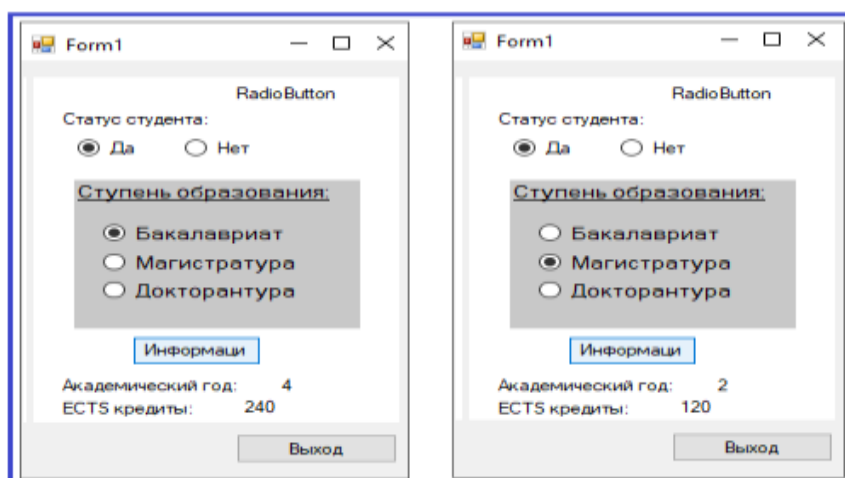


Рис.7.5

Программный код реализации указанного задания дан в следующем листинге:

```
// листинг--- CheckBox, RadioButton, Checked, CheckedChanged, SplitContainer --
using System;
using System.Drawing;
using System.Windows.Forms;
namespace WinFormCheckRadio
{
publicpartialclassForm1 : Form
{
public Form1()
{
InitializeComponent();
}
}
```

```
private void button2_Click(object sender, EventArgs e)
{
    Form2 f2 = new Form2();
    f2.Show();
}
private void splitContainer1_Panel1_Paint(object sender, PaintEventArgs e) {}

private void button1_Click(object sender, EventArgs e)
{
    Close();
}
// явление CheckedChanged – меняет положение элементов ---
private void checkBox1_CheckedChanged(object sender, EventArgs e)
{
    // включением checkBox-а появится панель
    if (checkBox1.Checked) // checked свойство checkBox-а для контроля положения
    {
        panel2.Visible = true;
    }
    else // выключением checkBox-а панель прячется
    {
        panel2.Visible = false;
    }
}
// подсчет часов
private void button4_Click(object sender, EventArgs e)
{
    int s = 0;
    if (checkBox2.Checked)
        s += 15;
    if (checkBox3.Checked)
        s += 15;
    if (checkBox5.Checked)
        s += 15;
    if (checkBox8.Checked)
        s += 30;

    label9.Text = s.ToString();
}
// код иллюстрации радио-кнопки
private void radioButton1_CheckedChanged(object sender, EventArgs e)
{
    if (radioButton1.Checked) // В radioButton вставляем маркер
    // появится панель
    {
        panel1.Visible = true;
    }
}
```

```
    }
else// удалением маркера из radioButton-а панель прячется
{
    panel1.Visible = false;
}
}
// клавиша „информация“ выводит результат ----
privatevoid button3_Click(object sender, EventArgs e)
{
if (radioButton2.Checked)// вставленный пример if...else if...else
{
    label12.Text = " 4";
    label13.Text = "240";
}
elseif (radioButton3.Checked)
{
    label12.Text = " 2";
    label13.Text = "120";
}
else
{
    label12.Text = " 3";
    label13.Text = "180";
}
}
// переходом из одного радио-бтона на другой очищаются данные
// старого результата
privatevoid radioButton2_CheckedChanged(object sender, EventArgs e)
{
    label12.Text = " ";
    label13.Text = " ";
}
privatevoid radioButton3_CheckedChanged(object sender, EventArgs e)
{
    label12.Text = " ";
    label13.Text = " ";
}
privatevoid radioButton5_CheckedChanged(object sender, EventArgs e)
{
    label12.Text = " ";
    label13.Text = " ";
}
}
}
```

## Лабораторная работа N8

### 2.3. Контейнерные элементы: GroupVox и TabControl

**Цель работы:** изучение визуальных контейнерных элементов GroupVox и TabControl.

#### ➤ Контейнер GroupVox

GroupVox является также групповой контрольной ячейкой Container класса с очерченной рамкой, в которую можно поместить несколько CheckBox-ов, RadioButton-ов или других визуальных элементов. Она объединяет однородные данные, текстовая идентификация которых осуществляется в левом верхнем углу рамки.

Контейнер GroupVox аналогичен рассмотренному ранее элементу Panel, на который помещались другие визуальные элементы. Здесь клавишей “экзамен” откроем окно Form2 и использованием элементов GroupVox, ChekVox, RadioButton решим задачу построения программного кода проекта задачи “Выбор экзаменационных предметов”. Вместе с этим должна быть учтена возможность работы на нескольких языках. Начальная форма приведена на рис.8.1:

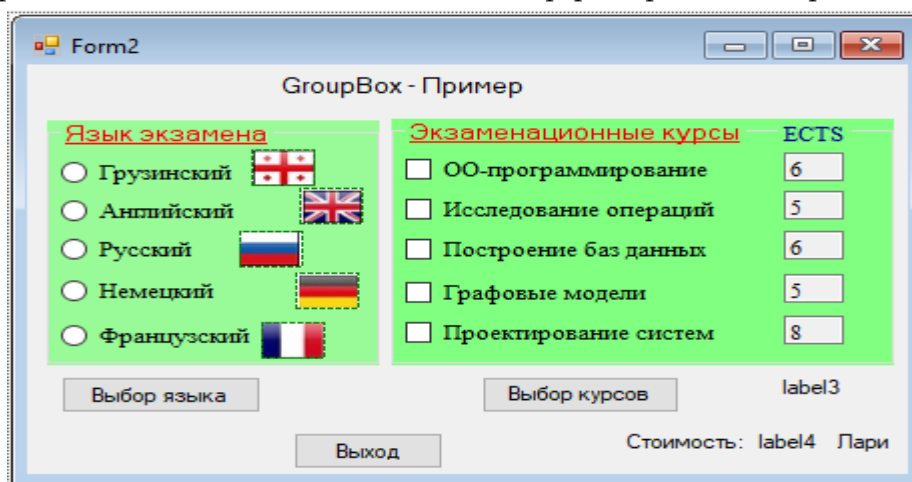


Рис. 8.1. Начальная форма интерфейса

На рисунке 8.2 для флажков (графический элемент) используется элемент управления PictureBox, для которого в файлах Resources (см. Solution Explorer) занесены заранее подготовленные соответствующие .jpg - файлы (Рис.8.2).

Элементы внутри GroupVox-а расположены в соответствии с наименованием данной группы. На рисунке показаны два групповых окна: “язык экзамена” и “экзаменационные курсы”.

После выбора нужного RadioButton-а языка клавишей “выбор языка” переходим на выбор экзаменационных предметов, активизируем необходимые нам CheckBox-ы.

Если язык поменялся на негрузинский, тогда меняются надписи экзаменационных предметов GroupVox-а и его компонент в соответствии с выбранным языком. Например, на рисунке 8.3 показаны варианты русского и английского языков.

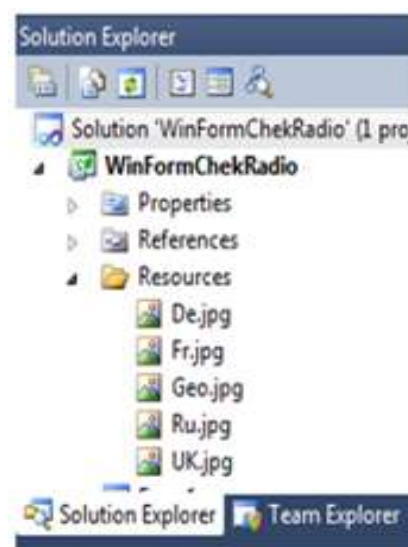


Рис. 8.2. Флажки стран

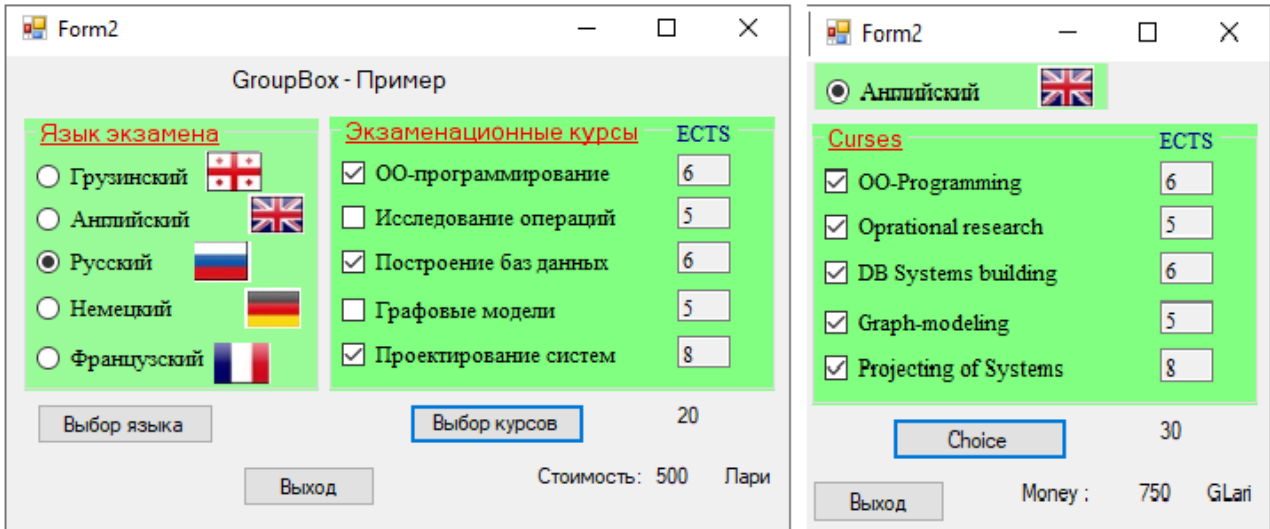


Рис. 8.3. Русский и английский варианты

В label13 выносятся суммарное число ECTS – кредитов предметов, выбранных студентом, а в label14 необходимую сумму оплаты. Есть возможность сокращения количества предметов (выключением checkBox-а) или добавлением нового (включением checkBox-а). В ячейках label13 и label14 моментально производится пересчет. Программный код соответствующих элементов и функционалов приведен в листинге:

```
// листинг ---- GroupBox, CheckBox, RadioButton, ImageBox ---
using System;
using System.Drawing;
using System.Windows.Forms;
namespace WinFormChekRadio
{
publicpartialclassForm2 : Form
{
    public Form2()
    {
        InitializeComponent();
    }
    privatevoid checkBox1_CheckedChanged(object sender, EventArgs e){ }
    privatevoid button1_Click(object sender, EventArgs e) //выбор грузинского языка
    {
        if (radioButton1.Checked)
        {
            groupBox2.Text = "საგამოცდო საგნები";
            button2.Text = "აირჩიეთ";
            checkBox1.Text="ოო-პროგრამირება";
            checkBox2.Text="ოპერაციათა კვლევა";
            checkBox3.Text="მონაცემთა ბაზების აგება";
            checkBox9.Text="გრაფული მოდელები";
            checkBox5.Text="სისტემების დაპროექტება";
            label15.Text = "თანხა .:";
            label16.Text = "ლარი";
        }
        elseif (radioButton2.Checked)// выбор английского языка
        {
            groupBox2.Text = "Curses";
        }
    }
}
```

```

        button2.Text = "Choice";
        checkBox1.Text="OO-Programming";
        checkBox2.Text="Oprational research";
        checkBox3.Text="DB Systems building";
        checkBox9.Text="Graph-modeling";
        checkBox5.Text="Projecting of Systems";
        label5.Text = "Money .:";
        label6.Text = "GLari";
    }
elseif(radioButton5.Checked)// выбор русского языка
    {
        groupBox2.Text = "Предметы";
        button2.Text = "Выбор";
        checkBox1.Text="OO-программирование";
        checkBox2.Text="Исследование операций";
        checkBox3.Text="Построение баз данных";
        checkBox9.Text="Графовые модели";
        checkBox5.Text="Проектирование систем";
        label5.Text = "Стоимость:";
        label6.Text = "GLari";
    }
else// выбор немецкого языка
    {
        groupBox2.Text = "Fachdisziplinen";
        button2.Text = "Wählen Sie bitte";
        checkBox1.Text = "OO-Programmierung";
        checkBox2.Text = "Operationsforschung";
        checkBox3.Text = "Entwicklung von Datenbanken";
        checkBox9.Text = "Graphische Modelen";
        checkBox5.Text = "Systementwurf";
        label5.Text = "Kosten .:";
        label6.Text = "GLari";
    }
}

privatevoid button2_Click(object sender, EventArgs e)
    {
        // кредиты по курам ----
        int s = 0, Sum=0 ;
        if (checkBox1.Checked) s += 6;
        if (checkBox2.Checked) s += 5;
        if (checkBox3.Checked) s += 6;
        if (checkBox9.Checked) s += 5;
        if (checkBox5.Checked) s += 8;
        // расчет суммы оплаты соответствующих кредитов
        Sum = s * 37.5;
        label3.Text = s.ToString();
        label9.Text = Sum.ToString();
    }
privatevoid button3_Click(object sender, EventArgs e)
    { Close(); }
}
}

```

➤ **Контейнер TabControl.**

Контейнерный элемент TabControl используется для организации в форме многостраничного диалога на основе применения принципа взаимно перекрывающихся страниц (Рис.8.4) - Аналогично страницам Excel (sheets). Добавление новой страницы осуществляется AddTab-ом, а стирание старого RemoveTab-а строчками из контекстного меню, которое выносится в правый угол нажатием на маленькую стрелку правой клавиши мыши. Запись имени каждой страницы осуществляется в Text Propertis-а соответствующей страницы.

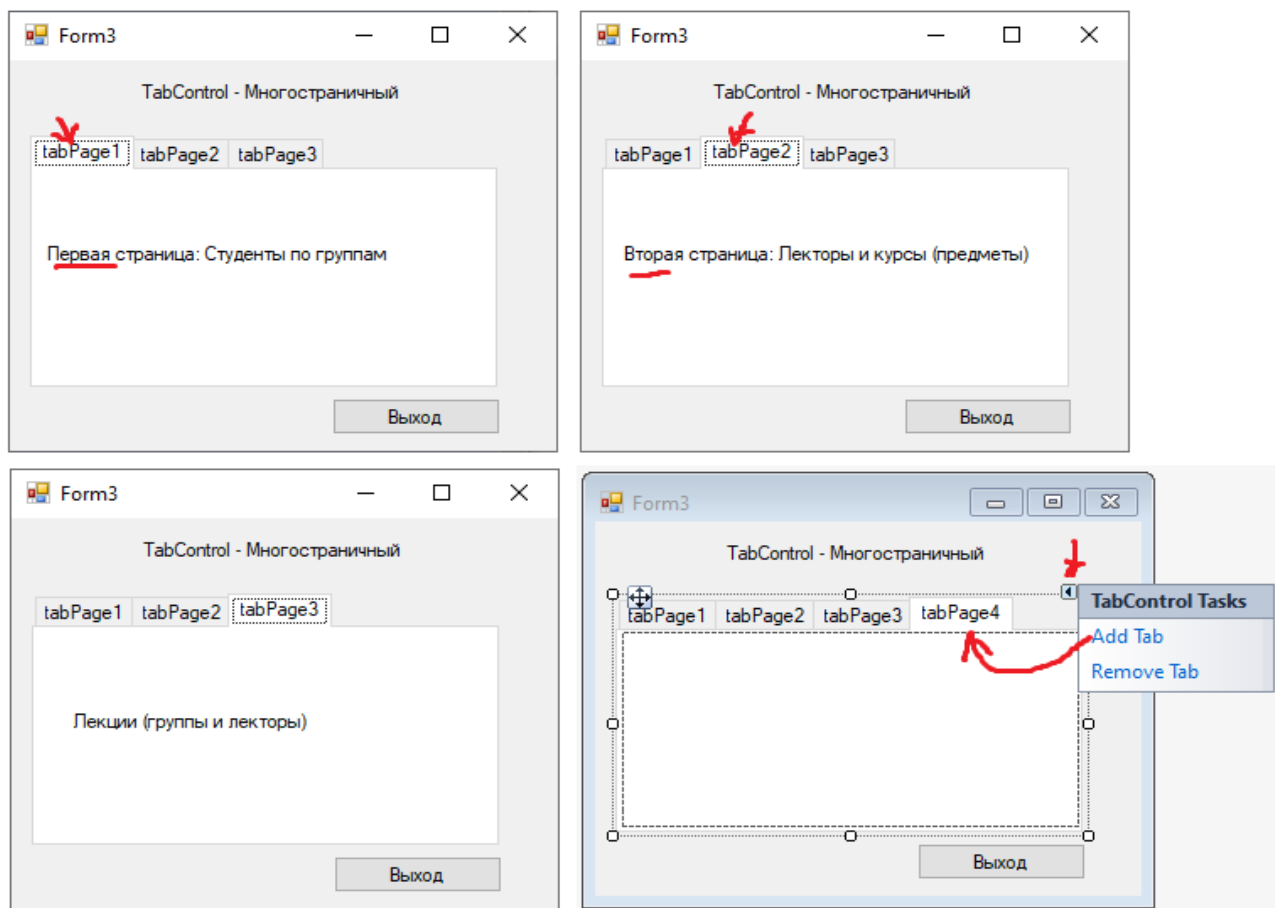


Рис. 8.4. Пример контейнера TabControl.

➤ **Самостоятельная работа:**

Постройте двух-панельную форму.

В первую вложите радиобутона с четырьмя элементами:

- 1) хлебодукты;
- 2) безалкогольные напитки;
- 3) молочная продукция;
- 4) кондитерские изделия.

На второй панели расположите 5 чекбоксов, в строках которых появится наименование конкретной продукции, выбранной клиентом из соответствующей категории.

Когда на левой панели изменится категория, соответственно на правой панели должно измениться наименование продукции.

- После выполнения задания сохраните результаты в eLearning-e.

Глава 3

Обработка данных строчного типа

Лабораторная работа №9

3.1. Данные String строчного типа и класс String

**Цель работы:** изучение принципов и методов обработки текстов и данных строчного типа.

Запоминание строк осуществляется String классом, синонимом которого является string тип данных. Объекты String класса, т.е. строки, обладают многими свойствами. Свойство Length используется для установления длины строки, или иначе говоря из скольких символов состоит строка.

**Задача 1:** установим на форме textBox1 (Рис.9.1-а) ячейку для занесения строки label2 – ячейку для вынесения значения длины строки. Клавишей “длина” должна определится длина строки и ее печать. Результат показан на рис.9.1-б.

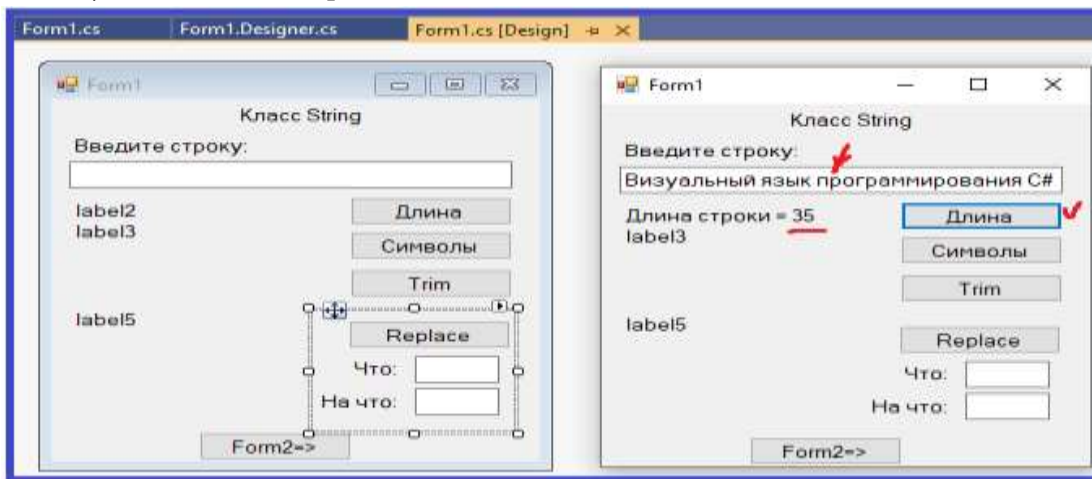


Рис.9.1. Пример использования свойства String.Length

Соответствующий программный код приведен в листинге:

```
// листинг --- String – строчный тип ---
private void button1_Click(object sender, EventArgs e)
{
    string Striqoni;
    Striqoni = textBox1.Text;
    label2.Text = "Длина строки = "+Striqoni.Length;
}

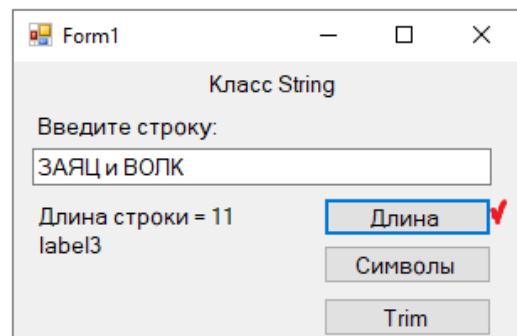
```

Таким образом, введенная строка была запомнена в переменной Striqoni string типа. Вычисление ее длины осуществляется с помощью свойства Striqoni.Length. В строке символы расположены на подобие массива. Символ является одним элементом и имеет собственный индекс (Таб.9.1)

З	А	Я	Ц		И		В	О	Л	К
0	1	2	3	4	5	6	7	8	9	10

Добавим форме Form1 одну клавишу “Символы” и привяжем ей код, который дан в следующем листинге:

Рис.9.2-а





```
// --- ЛИСТИНГ -- СИМВОЛЫ -----
private void button2_Click(object sender, EventArgs e)
{
    string Striqoni;
    char simbolo;// один символ
    int i;// индекс
    Striqoni = textBox1.Text;
    label2.Text="Символы";
    label3.Text="";
    for (i = 0; i < Striqoni.Length; i++)
    {
        simbolo = Striqoni[i];
        label3.Text += i.ToString()+" : "+ simbolo + "\n";
    }
}
```

Результат приведен на рис.9.2.

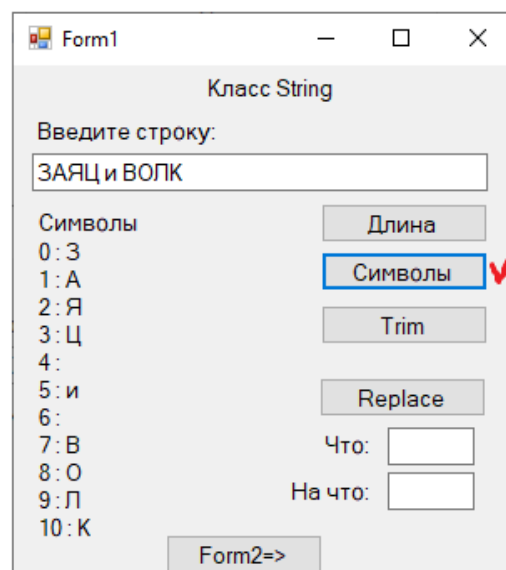


Рис.9.2-б

#### ➤ Методы обработки строки : Trim() и Replace()

Во время работы со строками часто встречаются пустые символы (пробелы), обработка которых (удаление, уменьшение и т.д.) необходима. Для этого в библиотеке есть несколько методов:

- Trim() – удаление ненужных символов с начала или с конца строки, в том числе и пробелов;
- Trim Start () - удаление ненужных символов с начала строки;
- Trim end () - удаление ненужных символов с конца строки;
- Replase () – метод удаления или замены нежелательных символов в середине строки.

На рис.9.3 показан пример Trim метода, фрагмент его программного кода описан в соответствующем листинге.

```
// листинг --- Trim -----
private void button3_Click(object sender, EventArgs e)
{
    string Striqoni, SuftaStriqoni;
    Striqoni = textBox1.Text;
    SuftaStriqoni = Striqoni.Trim(' ',';','#');
    label5.Text = SuftaStriqoni;
}
```

Как видно из результата, в начале и конце строки нет лишних символов. Осталось лишь ' ; ; ' и ' , , ' в середине строки. Теперь используем метод Replace() (см.листинг). Для этого в форму добавим эту клавишу (Рис.9.4).

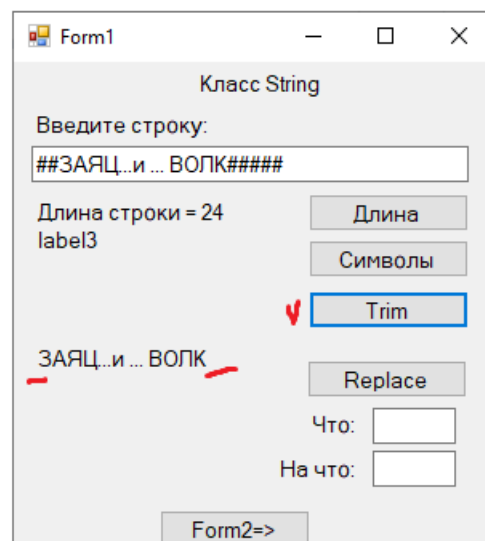


Рис.9.3.

// ----листинг --- Trim + Replace ----

```
private void button4_Click(object sender, EventArgs e)
{
    string Striqoni, SuftaStriqoni, Ra, Riti;
    Striqoni = textBox1.Text;
    Ra = textBox2.Text;
    Riti = textBox3.Text;
    SuftaStriqoni = Striqoni.Trim(' ', ';', '#');
    SuftaStriqoni = SuftaStriqoni.Replace(Ra, Riti);
    label5.Text = SuftaStriqoni;
}
}
```

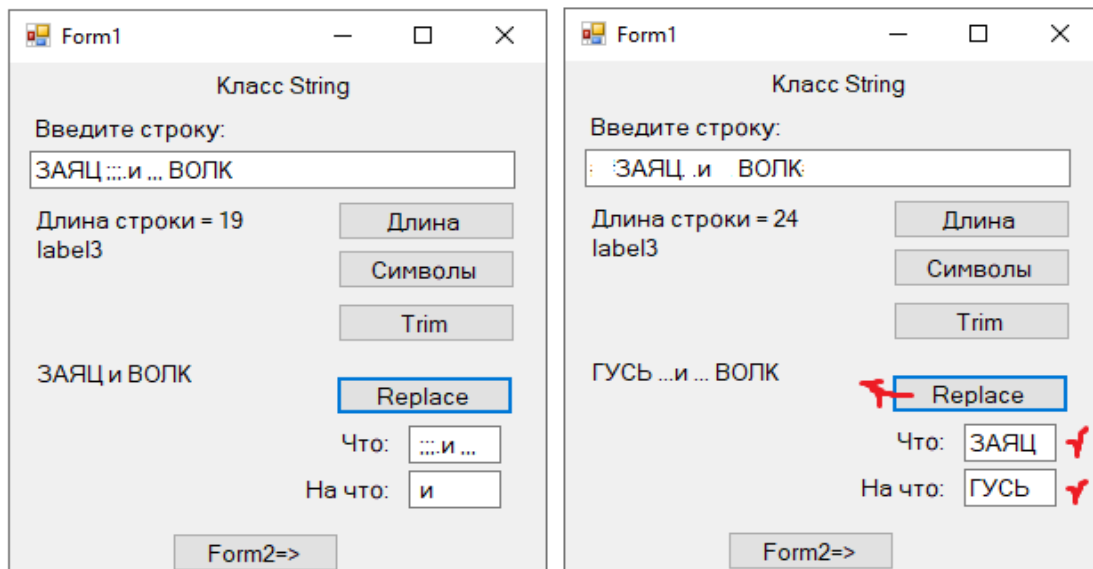


Рис.9.4. Добавление Replace метод

Метод Replace() требует применения дополнительно двух Textbox-ов, в которых указывается “что” надо заменить в строке и “на что” (Рис.9.5).

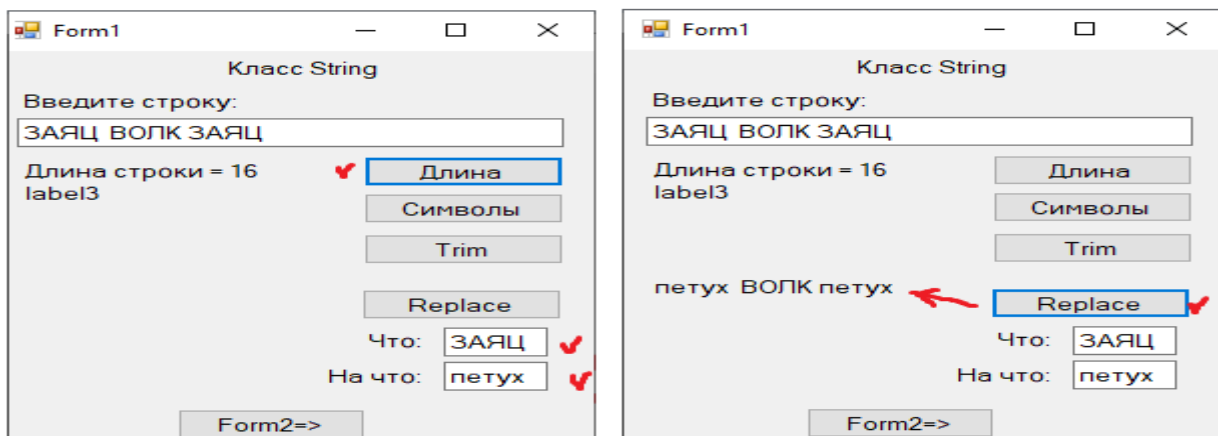


Рис.9.5. Примеры для Replace

➤ **Самостоятельная работа:**

Постройте C# код, который обрабатывает строку с вашим „именем, ... , ;;; и фамилией”, внесенную в текстбок: определите длину, разложите на символы, очистите от лишних символов и заново определите длину.

## Лабораторная работа N10

### 3.2. Методы поиска в строке:

#### IndexOf(), LastIndexOf() и IndexOfAny()

**Цель работы:** Изучение методов поиска IndexOf(), LastOf() и IndexOfAny() в строках и подстроках.

В форму (Рис.10.1) наряду с основной строкой (textBox1) поместим подстроку (textBox2) и привяжем к клавише “поиск” код (Рис.10.2), приведенный в листинге.

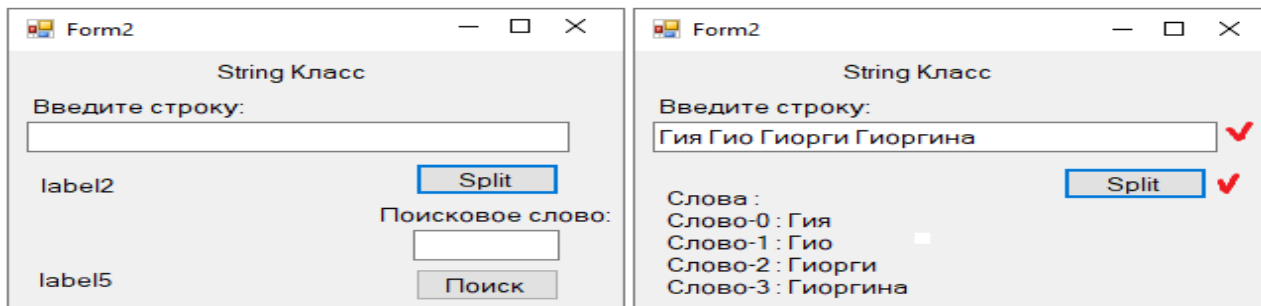


Рис.10.1. Split – деление строки по словам

```
// --- Листинг для Split клавиша -----
private void button1_Click(object sender, EventArgs e) // button1 Split
{
    string Striqoni = textBox1.Text; // Гия Гио Георги Георгина
    string[] Slovo;
    int i;
    Slovo = Striqoni.Split(' ');
    label2.Text = "Слова : " + "\n";
    for (i = 0; i < Slovo.Length; i++)
        label2.Text += "Слово-" + i + " : " + Slovo[i] + "\n";
}

```

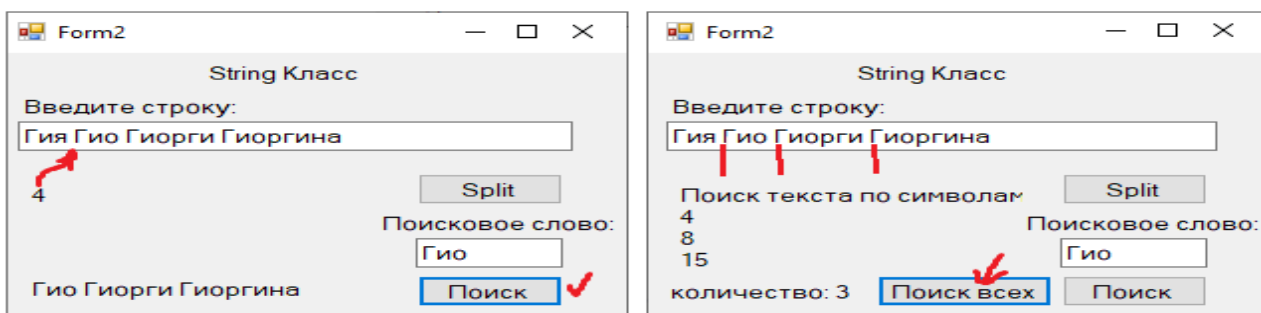


Рис.10.2. Поиск – позиции слова „Гю” в основную строку

```
// --- Листинг для клавиша Поиск – IndexOf() -----
private void button2_Click(object sender, EventArgs e) // button2 Поиск
{
    string Striqoni, Sazebni;
    int pozicia;
    label2.Text = "";
    Striqoni = textBox1.Text;
    Sazebni = textBox2.Text;
    pozicia = Striqoni.IndexOf(Sazebni); // !!!
    label2.Text += pozicia;
    label5.Text = Striqoni.Substring(pozicia);
}

```

На рисунке показано значение позиции местоположения (=4) искомой подстроки “Гео” (Sazebni – поисковое) в строке “Гиа Гео Гиорги Гиоргина “ (Striqoni - строка). Это первая найденная позиция искомой подстроки. Если значение позиции “0”, это указывает на начало основной строки, а если “-1”, значит такая подстрока не найдена.

Если необходимо зафиксировать все случаи существования искомой подстроки в основной строке, тогда алгоритм поиска приводится в листинге, а результат на рис.10.2 (правый).

```
// -- Листинг --- All IndexOf () ----
private void button3_Click(object sender, EventArgs e)
{
    string Striqoni, Sazebni;
    int pozicia; // позиция
    int zebnisDackeba = 0; // начало поиска
    int raod=0; // количество
    label2.Text = "";
    Striqoni = textBox1.Text; // строка
    Sazebni = textBox2.Text; // поисковая подстрока
    label2.Text = "Поиск текста по символам: " + "\n";
    do
    {
        pozicia = Striqoni.IndexOf(Sazebni, zebnisDackeba);
        zebnisDackeba = pozicia + 1;
        if (pozicia != -1)
        {
            label2.Text += pozicia + "\n";
            raod++;
        }
    }
    while (pozicia != -1)
        ;

    label5.Text = "количество: " + raod;
}
}
```

Как видно, внутри цикла do ... while осуществляется многократный поиск в основной строке искомой подстроки. Начальная искомая позиция 0, а следующая определяется после каждой вновь найденной позиции. Программа запоминает также количество найденных случаев.

➤ Рассмотрим пример метода **IndexOfAny()**, назначение которого поиск в строке первой подстроки с указанным символом. На рис.10.3 показан этот случай, а в листинге код программы для клавиши IndexOfAny.

```
// -- Листинг --- IndexOfAny() -- для button4 ----
private void button4_Click(object sender, EventArgs e)
{
    string Striqoni;
    int pozicia;
    label2.Text = ""; label6.Text = ""; label7.Text = "";
    Striqoni = textBox1.Text; // Анна, Вова, КоляЮ Ася, Соня - все студенты
    textBox2.Text = "А"; // сб "К"
    pozicia = Striqoni.IndexOfAny(new char[] { 'A', 'K', 'B' });
    label2.Text += pozicia;
    label5.Text = Striqoni.Substring(pozicia);
}
}
```

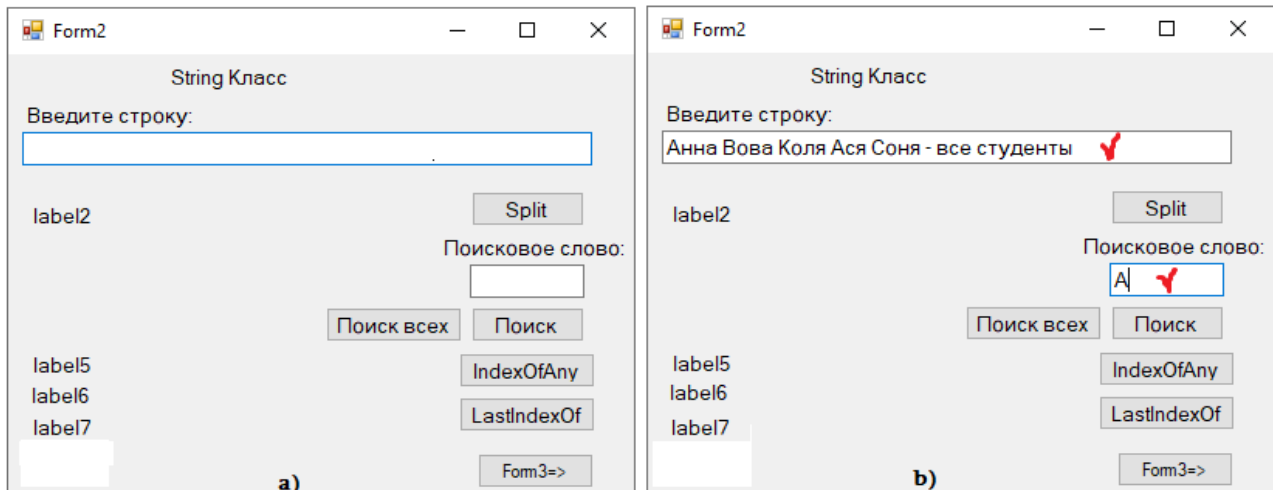


Рис.10.3. Результат применения метода IndexOfAny():  
 а) исходное состояние; б) Введены строка и символ (А)

➤ Теперь рассмотрим метод **LastIndexOf()**. Он в основной строке по указанному символу или слову ищет подстроку, но **с конца** (не первый слева). Соответствующий код программы дан в листинге:

```
// --- Листинг --- LastIndexOf() -- для button5 ----
private void button5_Click(object sender, EventArgs e)
{
    string Striqoni;
    int pozicia;
    label2.Text = ""; label5.Text = ""; label6.Text = ""; label7.Text = "";
    Striqoni = textBox1.Text;
    textBox2.Text = "А";
    pozicia = Striqoni.LastIndexOf("А");
    label6.Text += pozicia;
    label7.Text += Striqoni.Substring(pozicia);
}

```

В результате (Рис.10.4) видна разница между методами IndexOf Any() и LastIndexOf().

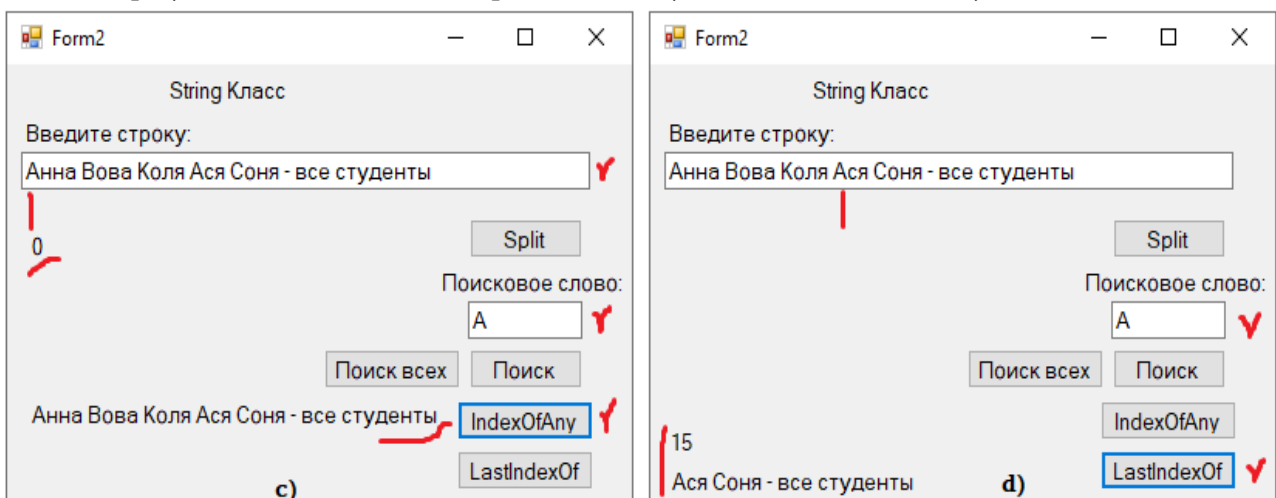


Рис.10.4. Результат применения метода LastIndexOf ():  
 с) Результат по IndexOfAny; д) Результат по LastIndexOf

## Лабораторная работа N11

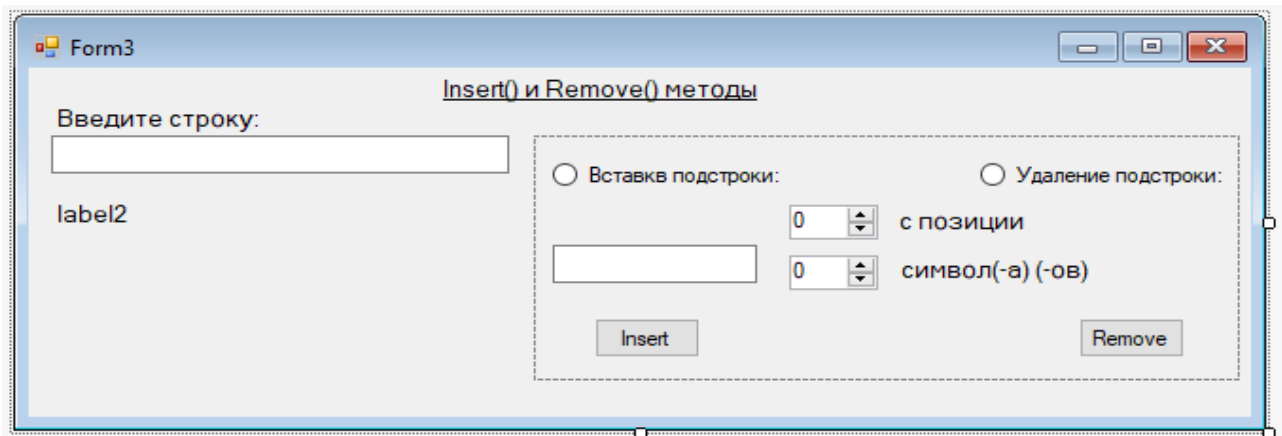
### 3.3. Работа со строками:

#### Insert(), Remove()

**Цель работы:** Изучение методов поиска символов Insert() и Remove() в строках и подстроках.

➤ **Метод Insert()** обеспечивает ввод одной строки в другую. Например, в основную строку (textBox1) введено некоторое предложение. В видимую строку на панели textBox2 вводится символ или цепочка символов, которая должна встать на позицию соответственно указанному числу (numericUpDown1) (Рис.11.1). В начале в Properties-е элемента numericUpDown1 Maximum=0, Minimum=0 и Value=0 т.е. в этом случае подстрока присоединится к основной только в начале. Если позицию установим на "4", тогда получим правильный результат.

Рис.11.1. Исходная форма



Программный код показан в листинге:

```
// --- Листинг --- Insert() -----
private void button1_Click(object sender, EventArgs e) // Insert()
{
    string Striqoni, qveStriqoni;
    Striqoni = textBox1.Text;
    qveStriqoni = textBox2.Text;
    label2.Text = Striqoni.Insert((int) numericUpDown1.Value, qveStriqoni);
}
}
```

Подготовка основной строки и подстроки для Insert() функции (рис.11.2).

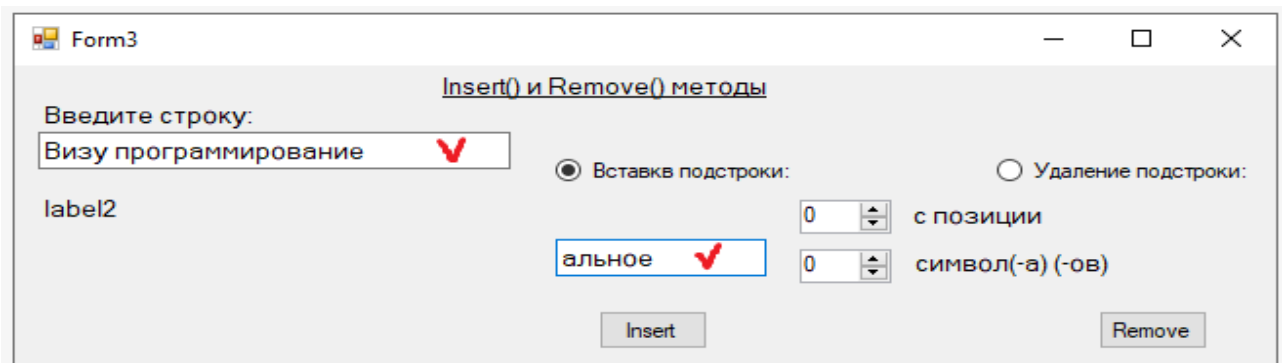


Рис.11.2. Строка и подстрока для вставки (с позиции "0")

Результаты (неправильный и правильный) показаны на рис.11.3-а и 11.3-б:

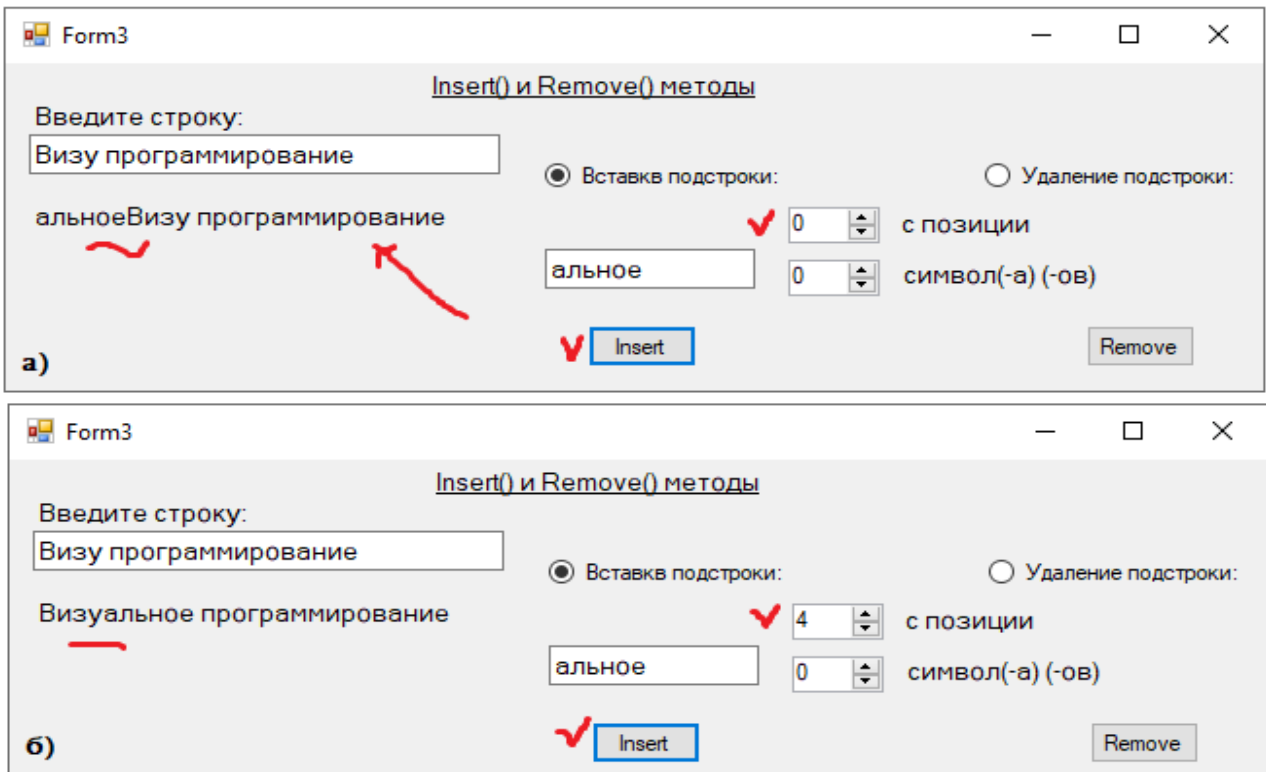


Рис.11.3. Результат применения Insert() метода  
а) неправильный и б) правильный

Здесь в label-е видно слово “Визуальное”, которое осуществилось Insert() методом. Если поменять позицию на “15”, а затем на “17”, то получим бессмысленный результаты). На большее число счетчик не увеличит значение, так как это значение длина основной строки.

Для того, чтобы можно было осуществить изменение основной строки и параметра ее длины, необходим программный код, который показан на в листинге.

```
//--- Листинг изменение длины основной строки ---
private void textBox1_TextChanged(object sender, EventArgs e)
{
    string Striqoni;
    Striqoni = textBox1.Text;
    numericUpDown1.Maximum = Striqoni.Length;
}
}
```

Как только в textBox-е начнется изменение текста основной строки, в тот же момент начнет работать этот программный код и изменится число, ограничивающее длину строки (Рис.11.4) в numericUpDown1.Maximum:

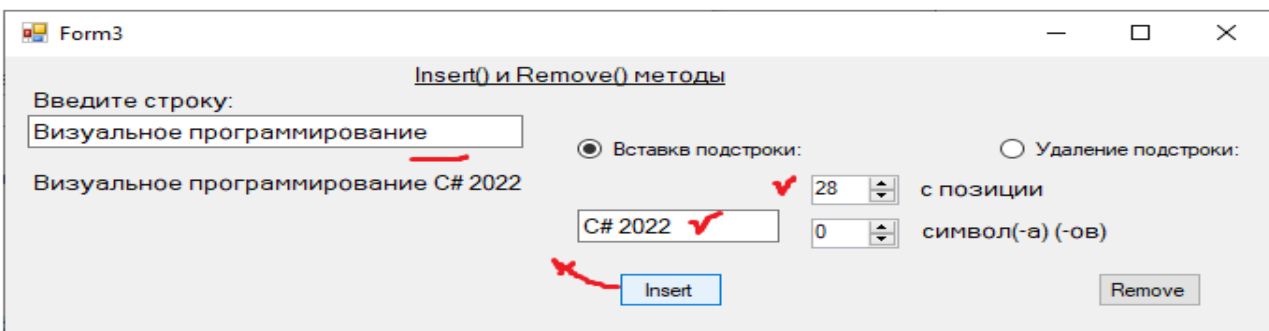


Рис.11.4. Результат применения Insert() метода

➤ **Метод Remove()** обеспечивает удаление из основной строки подстроки в соответствии с указанной позицией и количеством символов (Рис.11.1). Рассмотрим конкретный пример работы этого метода. Радиобутона “Удаление подстроки” заполним маркером, введем значение основного текста в textBox1() и определим соответствующие позиции удаляемой подстроки [начальная позиция и количество удаляемых символов] (Рис.11.5).

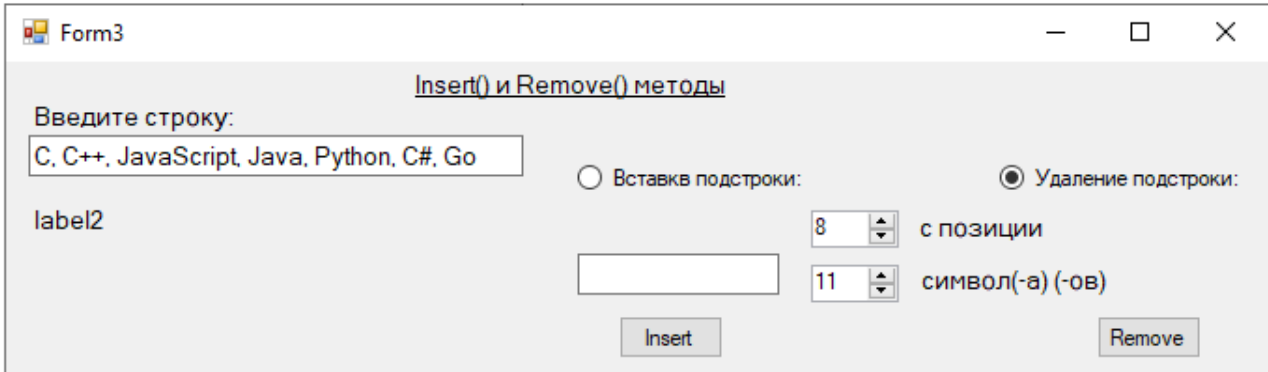


Рис.11.5. Удалить из основной строки Интерпретируемые языки программирования

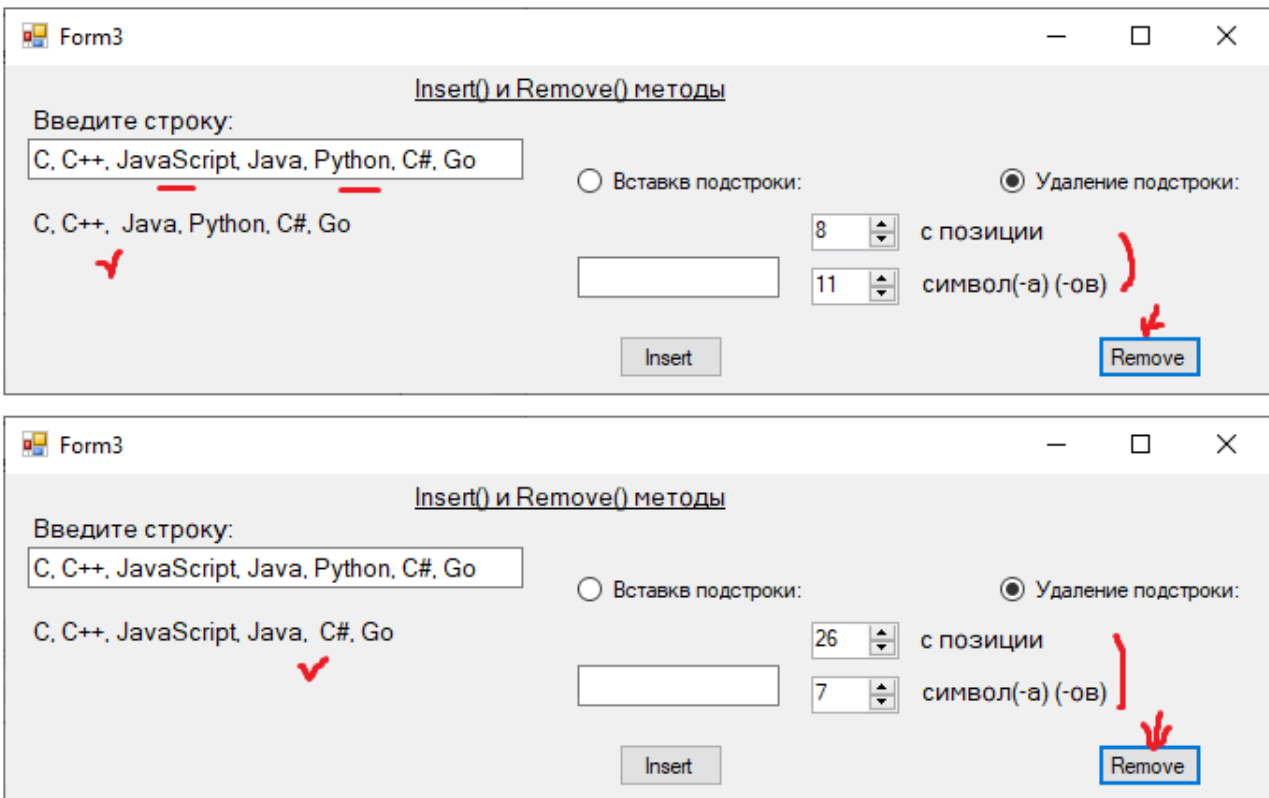


Рис.11.6. Результаты удаления

```
// ---- Листинг --- Remove() -----
private void button2_Click(object sender, EventArgs e)
{
    string Striqoni;
    Striqoni = textBox1.Text; // C, C++, JavaScript, Java, Python, C#, Go
    label2.Text = Striqoni.Remove((int)numericUpDown1.Value,
                                (int)numericUpDown2.Value);
}

```



```
private void numericUpDown1_ValueChanged(object sender, EventArgs e)
{
    string Striqoni = textBox1.Text;
    numericUpDown2.Maximum = Striqoni.Length - numericUpDown1.Value;
}
```

Результат показан на рис.11.6.

Назначением метода numericUpDown1\_ValueChanged() является контроль допустимых чисел в счетчиках “позиция” и “символы”. Длина нашей строки = 40. Например, если требуется удаление всех символов, с позиции” 26 (с Python), и укажем кол.-симв.=14, то получем результат (рис.11.7-а). Если в “позиция” запишем число 39, тогда в “символ” автоматически появится число=1 (рис.11.7-б). Дальше изменение числа в “позиция“ в сторону увеличения невозможно. Наоборот, возможно уменьшение.

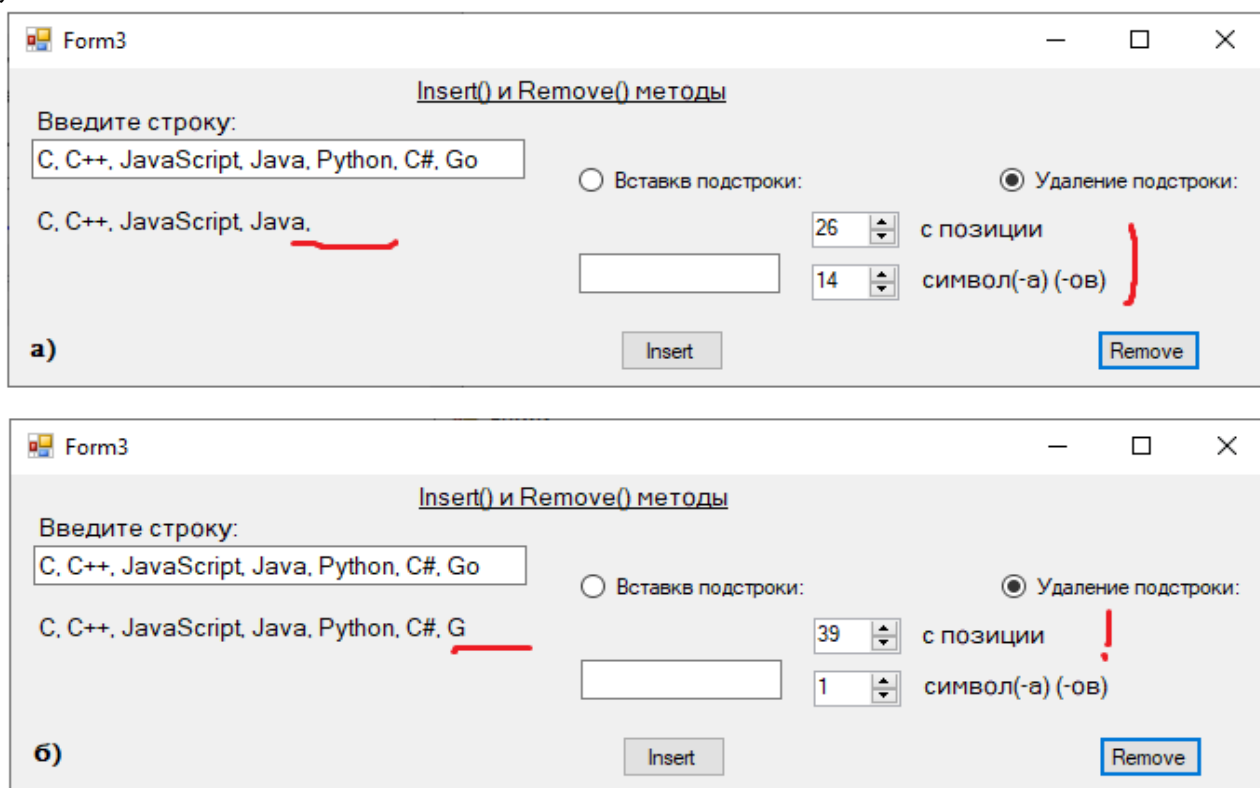


Рис.11.7. Автоматический контроль значений позиции и символов в счетчиках

➤ **Самостоятельная работа (для Гл.3):**

1) В основной строке (textBox1) запишите свое имя и фамилию, а в подстроке (textBox2) - отчество. Для button-а напишите C# код, создающий строку: «Имя — Отчество — Фамилия». Отобразить результаты в текстовом поле (Label);

2) Напишите код C#, который ищет указанное слово в предложение (текстовый абзац) и подсчитывает количество всех символов и количество гласных букв. Отобразить результаты в текстовых полях (Label 1 и 2);

3) Напишите код C#, который ищет указанное слово в текстовом файле (Notepad.txt или Word.docx) и подсчитывает, сколько раз это слово было найдено. Отобразить результаты в текстовом поле (Label)ю

## Глава 4

### Визуальные элементы управления

#### ListBox и ComboBox

Визуальные элементы ListBox и ComboBox значительно упрощают интерфейсы пользователей и делают их более гибкими. В данной главе мы рассмотрим эти элементы и иллюстративный пример разработки с их помощью программного продукта. На рис.12.0 показана обобщенная картина элементов ListBox и ComboBox, которая иллюстрирует различие между ними.



Рис.12.0. Окно Editor ListBox и ComboBox

**ListBox** осуществляет отображение своих записей на форме. Пользователь может многократно выбирать эти записи и использовать. Для таких задач используется цикл. Список записей может быть очень объемным и может не помещаться в окне, поэтому у ListBox-а есть автоматически работающий компонент ScrollBar, который устанавливается из Properties-а со значением true (рис.12.0).

**Combox** является полем, из которого выбирается нужное значение. Он на экране интерфейса занимает маленькое место и очень удобен для применения.

## Лабораторная работа N12

### 4.1. Визуальные элементы управления

#### ListBox, CheckedListBox

**Цель работы:** изучение синтаксиса применения элементов ListBox, CheckedListBox, свойства и методы в задачах программирования.

Ввод текстовых строк в ListBox осуществляется исходя из свойства **Properties->Items** или выбором в *правом верхнем углу рамки* ListBox-а посредством нажатия правой клавиши мыши **Edit Items**-а (Рис.12.1):

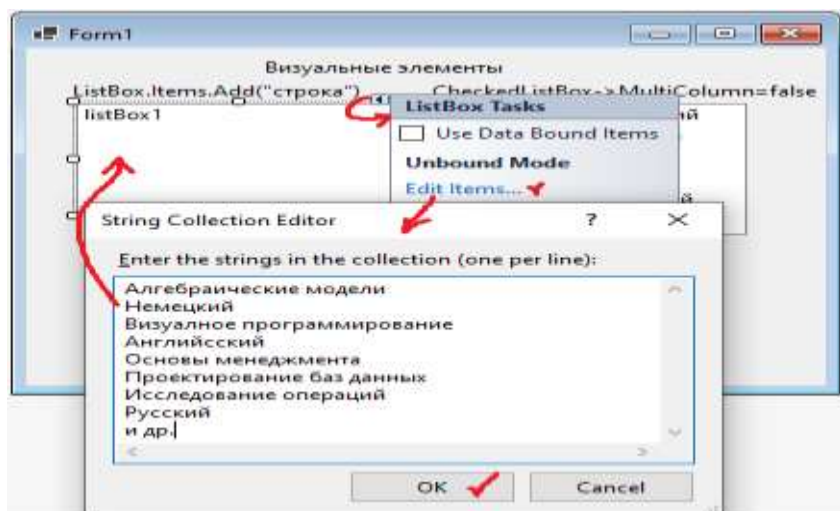


Рис.12.1. Окно Editor-а для ListBox

В обоих случаях выводится String Collection Editor, куда записываются данные.

Элемент CheckedListBox аналогичен ListBox-у, его строке впереди добавляется элемент checkBox. После внесения строки в редакторе Edit Items получаем рис.12.2.

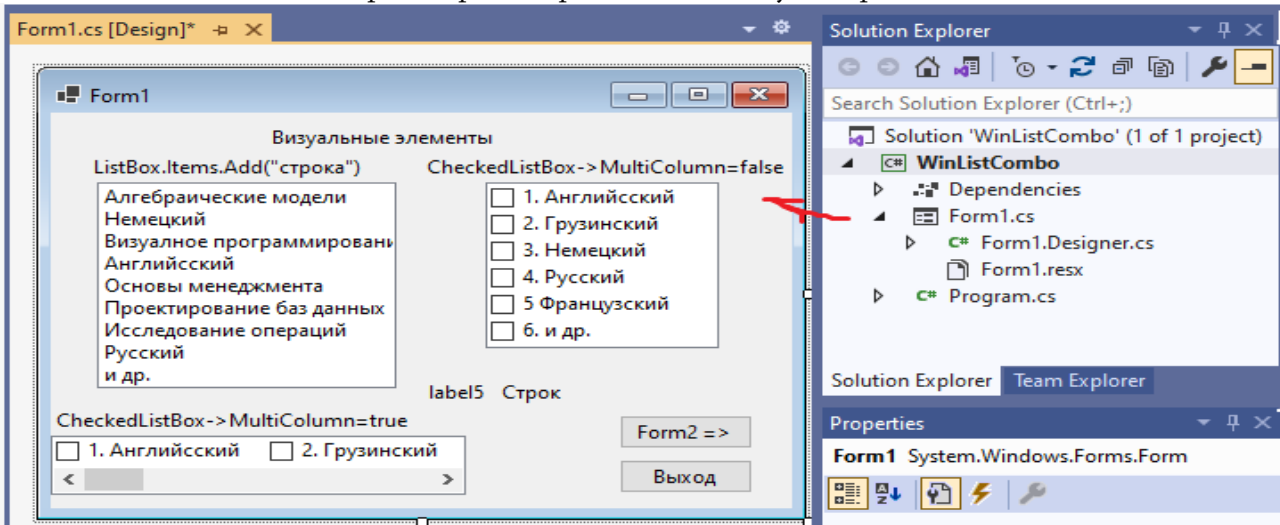


Рис.12.1. Form1 после ввода данных в Editor-е

Удалим у строк ListBox-а номера (если есть) и в его Properties-е установим Sorted->>true свойство. Строки будут упорядочены в соответствии с алфавитом (Рис.12.3).

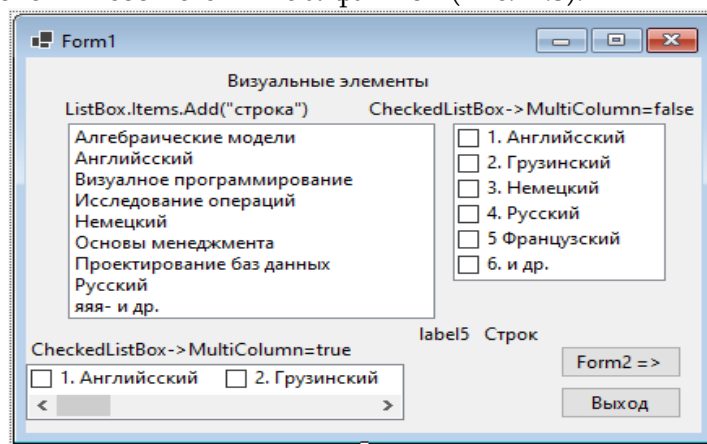


Рис.12.3

В CheckedListBox-е есть возможность **пометить** несколько строк. А в ListBox-е для этой цели необходим выбор SelectionMode->MultiExtended Properties-а. После этого возможно посредством применения клавиш Shift и Ctrl **пометить** одну или несколько строк (Рис.12.4).

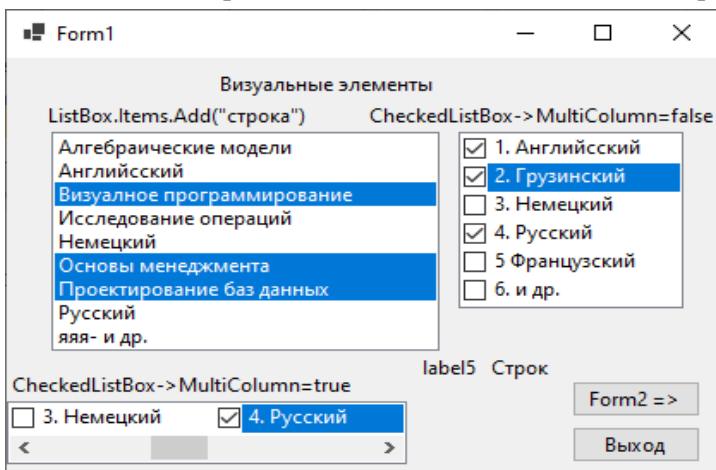


Рис.12.4

Теперь рассмотрим некоторые методы программной работы с текстовками.

- **Метод Add()** для свойства Item. Фрагмент программного кода, который осуществляет добавление строк в ListBox (Рис.12.5) показан в листинге:

```
// --- Листинг- Add()- метода свойства Item ListBox -a---
private void Form1_Load(object sender, EventArgs e)
{
    int st;
    /* listBox1.Items.Add("Проектирование баз данных");
    listBox1.Items.Add("Визуальное программирование");
    listBox1.Items.Add("Исследование операций");
    listBox1.Items.Add("Алгебраические модели");
    listBox1.Items.Add("Основы Менеджмента");          */
    listBox1.Items.Add("Китайский язык"); // Добавление в listBox1
    st = listBox1.Items.Count; // кол.строк
    label5.Text = st.ToString();
    checkedListBox1.Items.Add("Китайский"); // Добавление в checkedListBox1
}
```

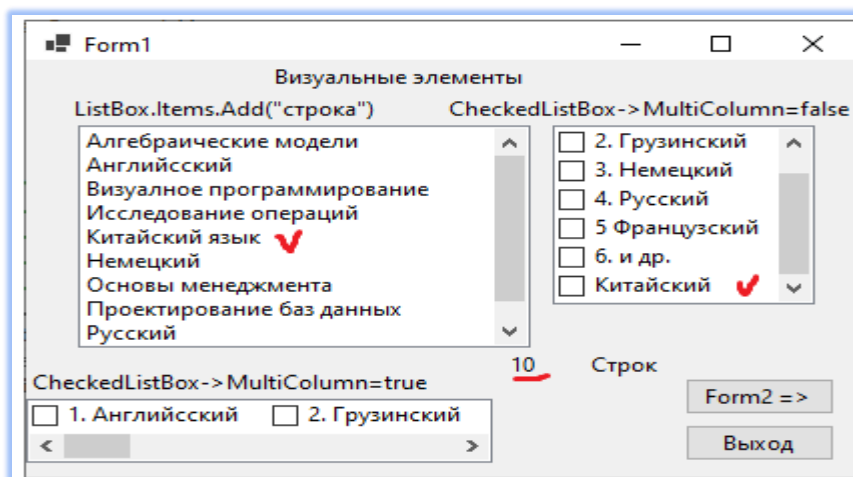


Рис.12.5. Результаты добавления строк

- **Методом Items.Count** определяется в ListBox-е количество строк: в label5 выносится 10 строк.

При работе с программным кодом для идентификации и выбора строк применяется свойство SelectedItem / SelectedItems и SelectedIndex / SelectedIndices. В последующем листинге показан фрагмент работы со строками ListBox-а (на Form2, рис.12.6):

```
// -- Листинг – SelectedItem, SelectedIndex, Items[index] -----
private void button1_Click(object sender, EventArgs e)
{
    int st;
    label12.Text = "Кол.строк =" + listBox1.Items.Count;
    label13.Text = "Выбранная строка =" + listBox1.SelectedItem;
    label14.Text = "Выбранной строки No =" + (listBox1.SelectedIndex + 1).ToString();
    label15.Text = "Все строки:" + "\n-----\n";
    for (st = 0; st < listBox1.Items.Count; st++)
        label15.Text += listBox1.Items[st] + "\n";

    label16.Visible = true; // Элемент появится на экране
    label17.Visible = true;
    label18.Visible = false; // Элемент не виден на экране
}
```

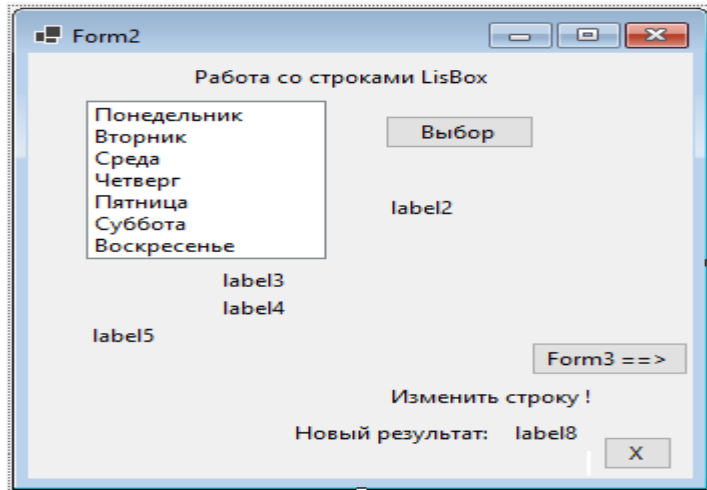


Рис.12.6. Интерфейс для работы со строками ListBox-а

Результаты после использования клавиши “выбор” показан на рис.12.7.

При изменении строк в ListBox-е автоматически должно исправиться выводимое новое значение результата (которое должно записаться в label8). с этой целью должно быть подготовлено событие listBox1\_SelectedIndexChanged, которое автоматически осуществит изменение. Его листинг приведен ниже, а событие для LitBox-а строится через его Properties (рис.12.8). Листинг события записывается в теле программы.

//---Листинг --- код события для изьенения строки в ListBox1 -----

```
private void listBox1_SelectedIndexChanged(object sender, EventArgs e)
{
    label8.Visible = true;
    label8.Text = " " + listBox1.SelectedItem;
}
```

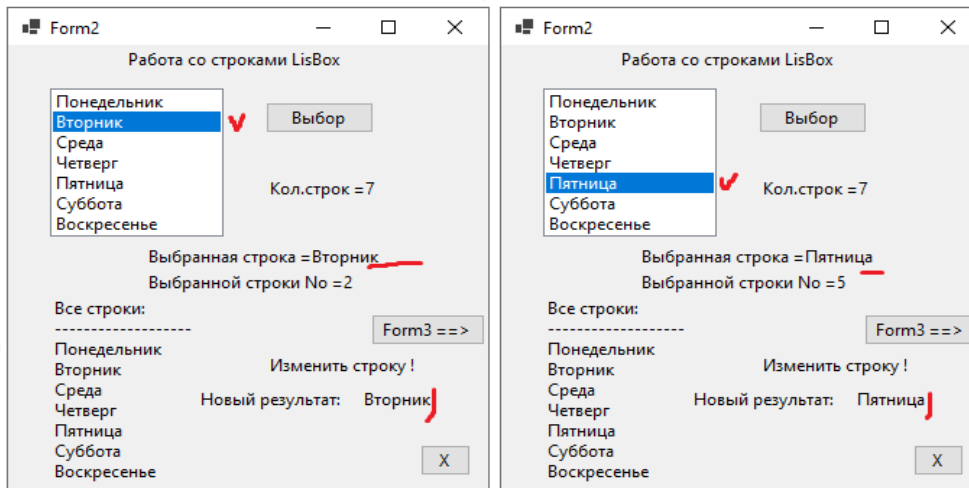


Рис.12.7. Автоматическое изменение строки

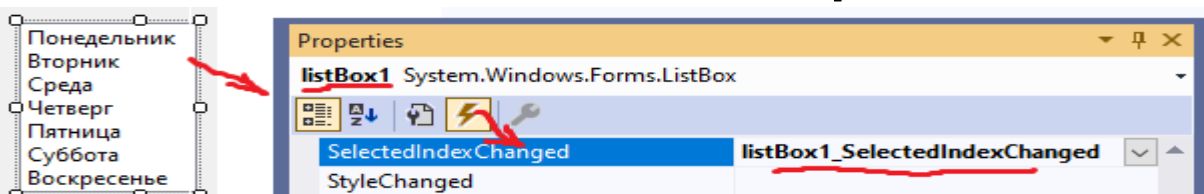


Рис.12.8. Событие для ListBox-а

**Задача 12.1:** запрограммируем для ListBox-а три метода: для ввода новой строки, вычеркивание ненужной строки и изменение значения строки. Возможно применение Insert() и Remove() методов. Для отображения результатов создадим Form3 (Рис.12.9-а). Из Form2 переходим на Form3 с помощью Button-а.

➤ Для клавиши “Добавление”, которая добавляет новую строку в начало, конец или перед указанной строкой ListBox-а. Код будет иметь вид, показанный в листинге (Рис.12.9-б).

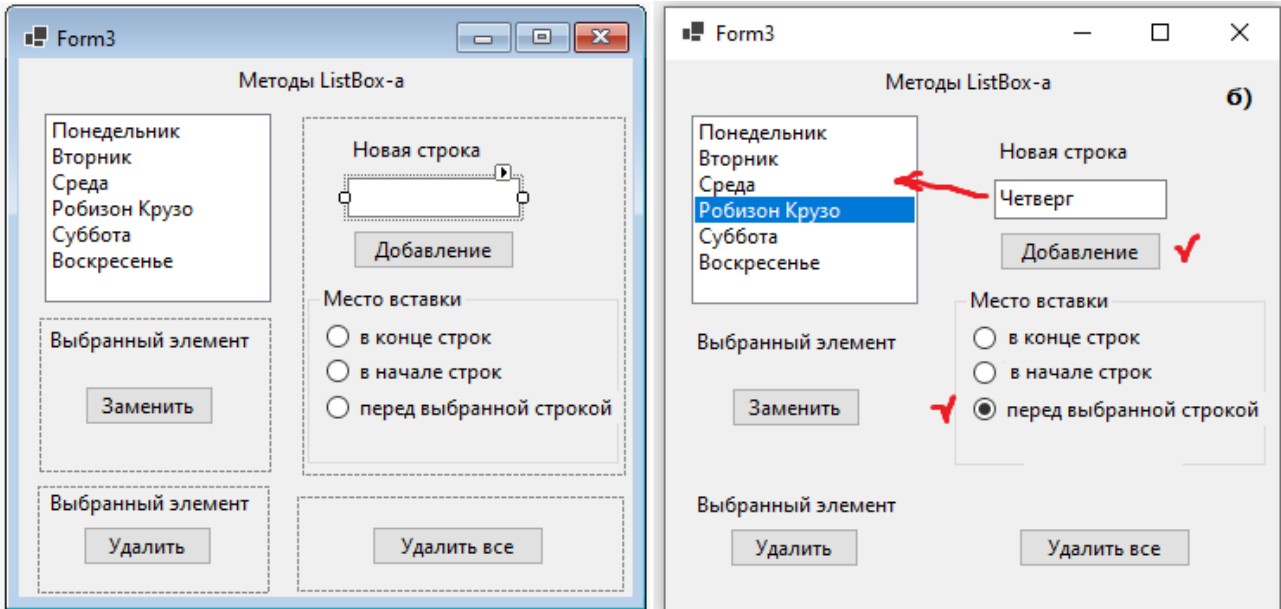


Рис.12.9

```
//--- Листинг – Insert() метод для ListBox1 -----
private void button1_Click(object sender, EventArgs e) // Բաձմաճոյձ
{
    if (textBox1.Text == "")
        return;

    if (radioButton2.Checked)
        listBox1.Items.Insert(0, textBox1.Text);
    elseif (radioButton3.Checked && listBox1.SelectedIndex != -1)
        listBox1.Items.Insert(listBox1.SelectedIndex, textBox1.Text);
    else
        listBox1.Items.Add(textBox1.Text);

    textBox1.Text = "";
}
}
```

После добавления новой строки (например, “Четверг”) listBox1 будет иметь вид, показанный на рис.12.10.

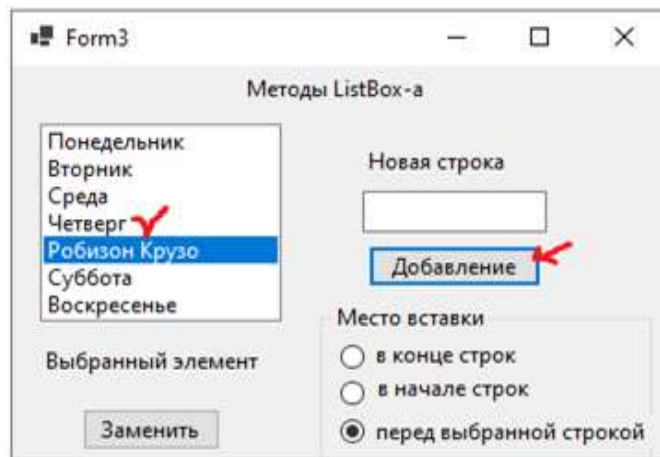


Рис.12.10

➤ Код клавиши “Стереть”, которая удалит выбранную строку из ListBox-а, показан в следующем листинге:

```
//--- Листинг метода удаления строки RemoveAt() для ListBox -а -----
private void button2_Click(object sender, EventArgs e) // стирание выбранного
{
    int st = listBox1.SelectedIndex;
    if (st != -1)
        listBox1.Items.RemoveAt(st);
}
```

В ListBox-е мы добавили фиктивную строку (“гхгхгх”) для эксперимента. Выберем эту строку и задействуем клавишу “Стереть”. В результате в листбоксе исчезнет данная строка.

Здесь работал Items.RemoveAt(st) метод, который SelectedIndex-ом передал номер стираемой строки. Если строка не указана, SelectedIndex возвращает значение „-1”. Результат показан на Рис.12.11.

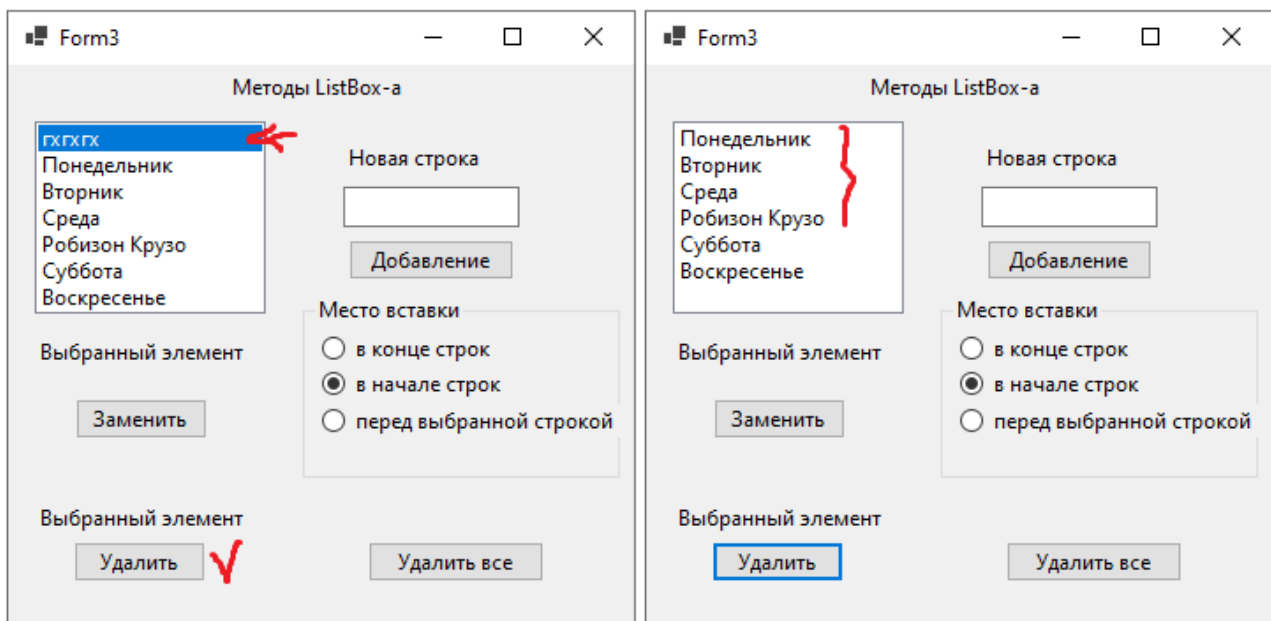


Рис.12.11. Удаление помеченной строки

➤ Применение клавиши “Стереть все” опасно, так как можно случайно потерять данные. Эта клавиша должна содержать дополнительное предупреждение для пользователя. И если от него получает “добро” на удаление всех строк, только тогда очистит все записи листбокса. В листинге показан фрагмент программного кода „Стереть все”:

```
//---Листинг -- ListBox.Items.Clear() метод -----
private void button3_Click(object sender, EventArgs e) // ყველასწამლა
{
    DialogResult all = MessageBox.Show("действительно все стереть ?",
        "предупреждение", MessageBoxButtons.YesNoCancel, MessageBoxIcon.Question);

    if (all == DialogResult.Yes)
    {
        label5.Text = "все стирается !";
        listBox1.Items.Clear();
    }
    else
        if (all == DialogResult.No)
```

```

label15.Text = "не все стирается!";
else
label15.Text = "Cancel-os !";
}

```

В нашем случае в коде применен метод Show(параметры) класса MessageBox с клавишами "Yes/No/Cancel" и с предупреждающим сообщением (Рис.12.12).

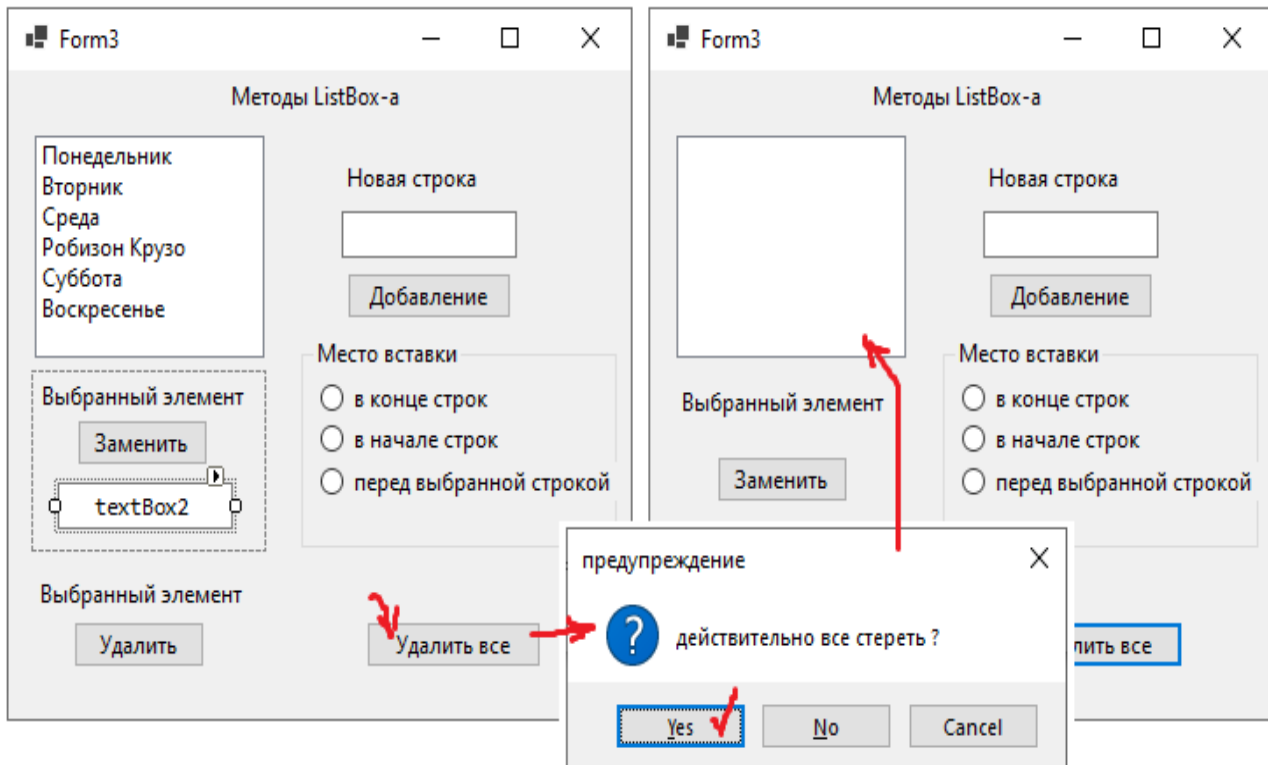


Рис.12.12. Результат: „Удаление всех строк“ - (Yes)

Рассмотрим строку кода:

```

DialogResult all = MessageBox.Show("Действительно все стереть ?",
    "предупреждение", MessageBoxButtons.YesNoCancel, MessageBoxIcon.Question);

```

*DialogResult* есть тип (enum) перечисления System.Windows.Forms сферы: `public enum DialogResult`, который определяет значение идентификатора, возвращенного из окна. **all**-у присваивается это значение, которое в дальнейшем применяется в **if** блоке.

*MessageBoxButtons* дает константы enum типа, которые определяют в окне MessageBox-а выносимые клавиши ( у нас **Yes, No, Cancel**).

*MessageBoxIcon* – константа enum типа, которая отображает сообщение. Например, `MessageBoxIcon.Question` есть символ знака „ ? “ белого цвета, помещенный в голубой круг.

Для того, чтобы *позметить* несколько строк ListBox-а, как было отмечено выше, в его свойство `SelectionMode Properties`-а помещаем значение `MultiExtended`-а.

При открытии нового листбокса в нем установлен одностроковый режим: `SelectionMode="One"` . Свойства `SelectedIndices` и `SelectedItems` содержат соответствующие номера или записи выбранных строк.

➤ Клавиша **“Заменить”** выбранную строку (рис.12.12). Замена выбранного элемента новой строкой выполняется с помощью следующего кода:



```
// --- Листинг – Замена строки для ListBox-a ---
private void button4_Click(object sender, EventArgs e)
{
    int st;
    if (textBox2.Text != "" && listBox1.SelectedIndex != -1)
    {
        st = listBox1.SelectedIndex;
        listBox1.Items.RemoveAt(st);
        listBox1.Items.Insert(st, textBox2.Text);
        textBox2.Text = "";
    }
}
```

Результат показан на рис. 12.13.

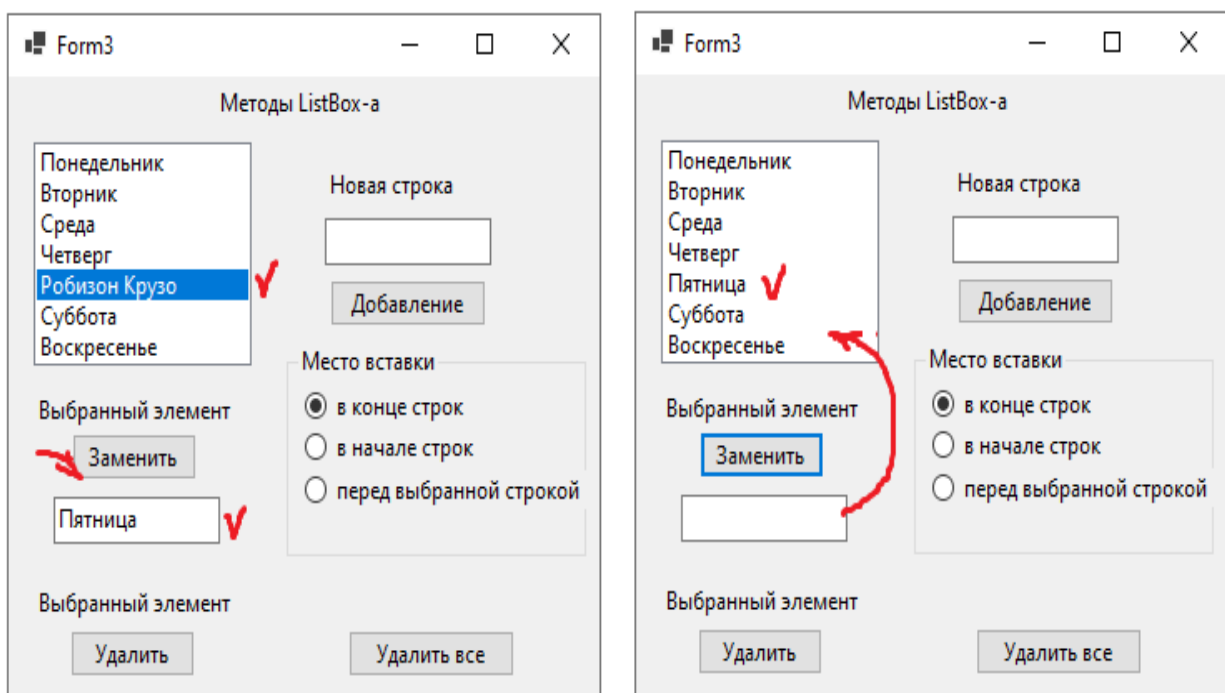


Рис.12.13. Пример замены строки „Робизон Крузо“ на „Пятницу“

➤ **Самостоятельная работа:**

**Задача 12.2:** построим код, с помощью которого возможно пометить несколько строк Listbox1-a и их одновременный перенос (Copy) в Listbox2. Одновременно должна быть возможность возвращения строки из Listbox2 в Listbox1. Код также должен содержать переноса строк (Move) между текстбоксами и полное удаление (DeleteAll) строк текстбокса.

## Лабораторная работа N13

### 4. 2. Визуальный элемент управления ComboBox и его свойство DropDownStyle

**Цель работы:** изучение функционирования визуального элемента управления ComboBox-а.

ComboBox намного упрощает интерфейсы пользователя и делает их более гибкими. Здесь рассмотрим примеры создания с его помощью программных проектов. На рис.13.1 пример, иллюстрирующий применение ComboBox-а.

Визуальный элемент ComboBox-а является симбиозом элементов ListBox-а и TextBox-а. Он использует показатели обоих элементов, которые рассмотрены выше. В тоже время форма его представления отличается от обеих и эффективна (занимает мало места на форме и использует механизм выбора). На рисунке показаны три варианта ComboBox-а. Его тип в Properties-е указывается одним из значений свойства *DropDownStyle*.

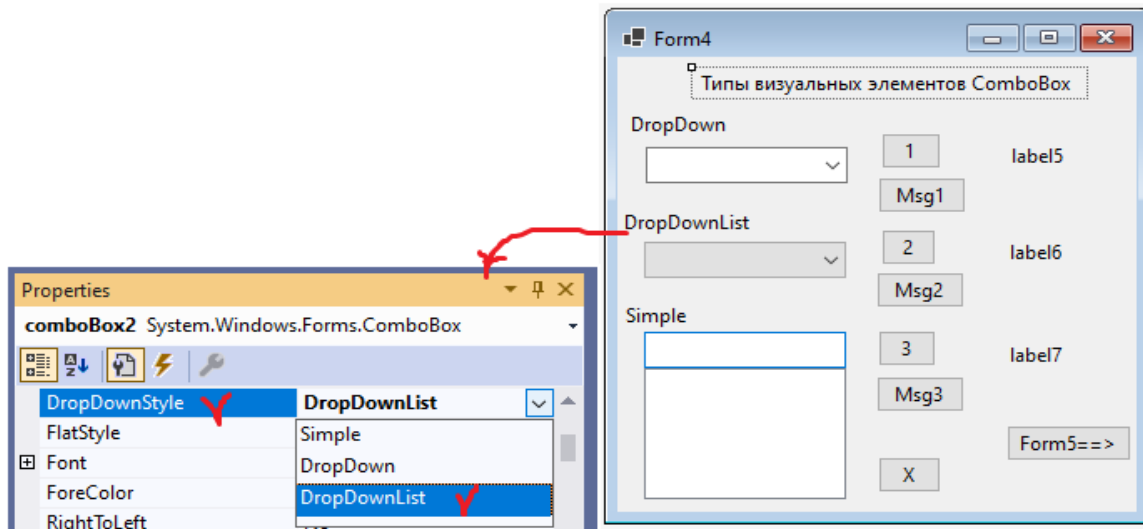


Рис.13.1. Типы ComboBox

**DropDown** – стандартно установлено это значение. С помощью треугольной стрелки появляется аналогично ListBox-у список строк, или в текстовое поле вводится строка также как в TextBox-е.

- **DropDownList** – отключает возможность TextBox-а, т.е. ввод текста исключается. Комбобокс работает в режиме листбокса и как текспбкс занимает мало места.
- **Simple** – список строк всегда открыт. Возможен как выбор строки, так и внесение новой.

Замечание: для ComboBox-а не применяется свойство SelectionMode.

**Задача 13.1:** построим форму, показанную на рис.13.1 и для иллюстрации трех различных типов комбобокса задействуем клавиши 1, 2, 3. В следующем листинге приведен программный код. Для иллюстрации значения выбранной строки (comboBox.SelectedItem) и ее индекса (номера) (comboBox.SelectedIndex), соответствующего комбобокса, введем в MsgBox-окно дополнительное сообщение.

Для каждой клавиши формы в Properties-е установим одно из значений свойства DropDown Style: 1- DropDawn, 2- DropDawnList или 3-Simple.

В Form4\_Load(object...) программно запишутся все три строки ComboBox-а, например C++, C#, F# и т.д. (нельзя применять редактор Edit Items).

//--- Листинг для типов ComboBox и свойства DropDownStyle ---

```
using System;
using System.Drawing;
using System.Windows.Forms;

namespace WinListCombo
{
    public partial class Form5 : Form
    {
        public Form4()
        {
            InitializeComponent();
        }

        private void Form4_Load(object sender, EventArgs e)
        {
            comboBox1.Items.Add("C++");
            comboBox1.Items.Add("C#");
            comboBox1.Items.Add("J++");
            comboBox1.Items.Add("F#");
            comboBox2.Items.Add("C++");
            comboBox2.Items.Add("C#");
            comboBox2.Items.Add("J++");
            comboBox2.Items.Add("F#");
            comboBox3.Items.Add("C++");
            comboBox3.Items.Add("C#");
            comboBox3.Items.Add("J++");
            comboBox3.Items.Add("F#");
        }

        private void button1_Click(object sender, EventArgs e)
        {
            label5.Text="Выбран: " + comboBox1.Text;
        }

        private void button2_Click(object sender, EventArgs e)
        {
            label6.Text = " Выбран: " + comboBox2.SelectedItem;
        }

        private void button3_Click(object sender, EventArgs e)
        {
            label7.Text = " Выбран: " + comboBox3.Text;
        }

        private void button6_Click(object sender, EventArgs e)
        {
            int selectedIndex = comboBox1.SelectedIndex;
            Object selectedItem = comboBox1.SelectedItem;
            MessageBox.Show("Selected Item Text: " + selectedItem + "\n" +
                "Index: " + selectedIndex.ToString());
        }

        private void button7_Click(object sender, EventArgs e)
        {

```

```

{
    int selectedIndex = comboBox2.SelectedIndex;
    Object selectedItem = comboBox2.SelectedItem;
    MessageBox.Show("Selected Item Text: " + selectedItem + "\n" +
        "Index: " + selectedIndex.ToString());
}
private void button8_Click(object sender, EventArgs e)
{
    int selectedIndex = comboBox3.SelectedIndex;
    Object selectedItem = comboBox3.SelectedItem;
    MessageBox.Show("Selected Item Text: " + selectedItem + "\n" +
        "Index: " + selectedIndex.ToString());
}

private void button4_Click(object sender, EventArgs e)
{
    Form5 f5 = new Form5();
    f5.ShowDialog();
}

private void button5_Click(object sender, EventArgs e)
{
    Close();
}
}
}

```

Результат работы программы показан на рис.13.2/

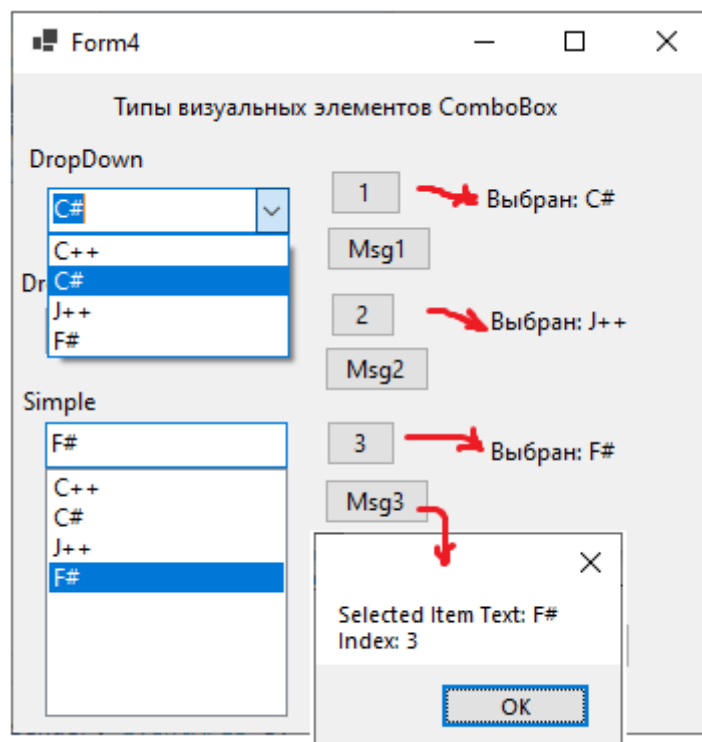


Рис.13..2.

## Лабораторная работа N14

### 4.3. Построение информационной системы „Университеты“

На базе пройденного материала построим новый программный проект, в котором будут использованы свойства и методы ComboBox класса.

**Задача 14.1:** создадим программную аппликацию “Университеты”, которая дает информацию о их факультетах и специальностях. Пользователь выбирает нужные ему данные в трехуровневой системе ComboBox-ов (Рис.14.1).

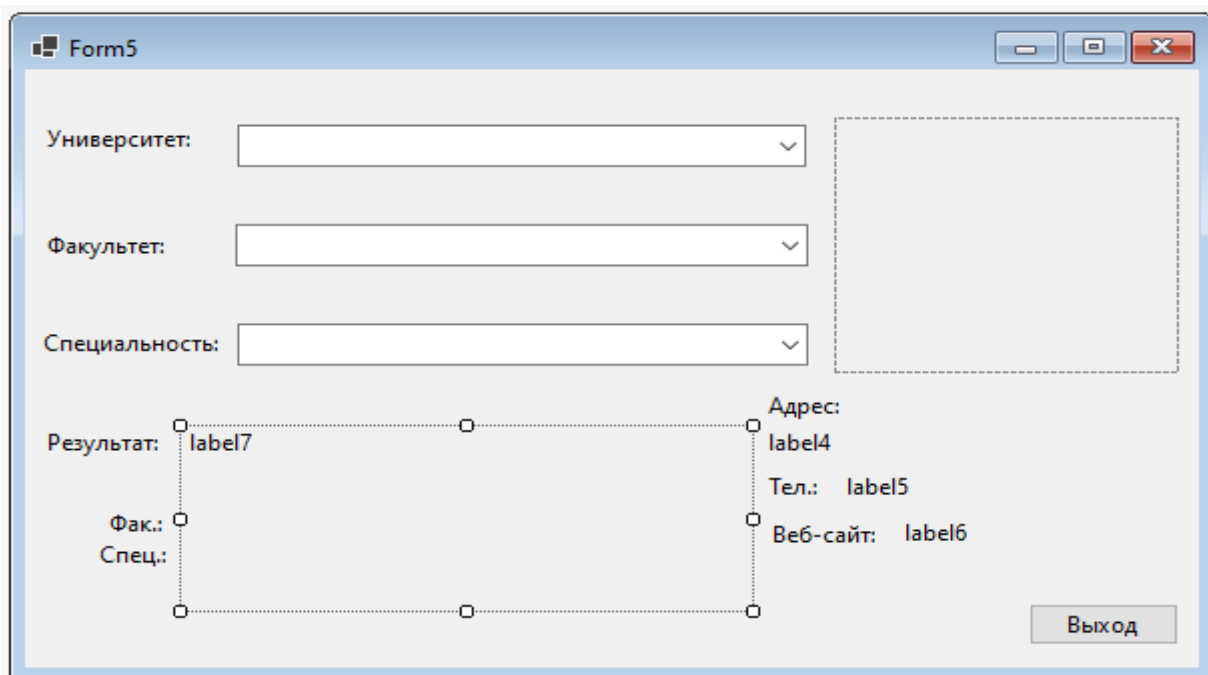


Рис.14.1

Результат отображается в текстовом поле label7. Изменение для выполнения нового запроса возможно на всех трех уровнях. Иерархическая модель, программная реализация которой возможна с помощью визуальных элементов и вложенными программными switch() переключателями, показана на рис.14.2.

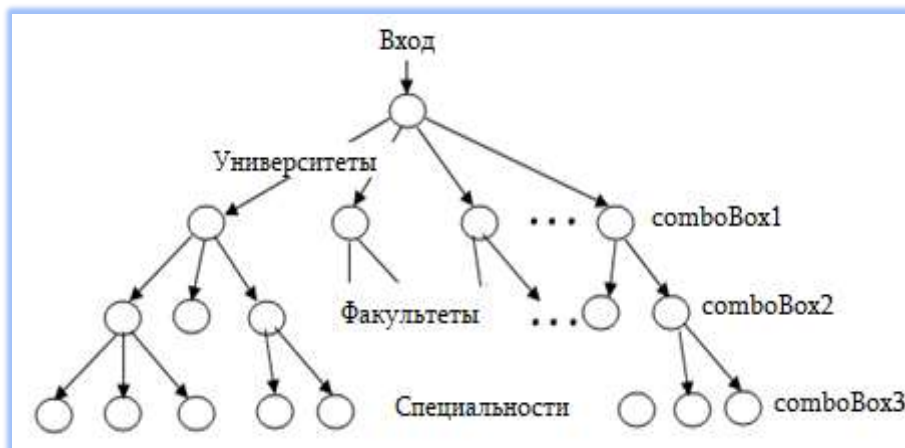


Рис.14.2. Иерархическая модель

На форме будет вынесена также графическая и текстовая (адрес, телефон, веб-страница) информация, соответственно с помощью pictureBox1 и трех label-ов. Их изменение осуществляется в соответствии с выбранным значением comboBox1. В следующем листинге приведен фрагмент реализации кода этой программы:

// --- Листинг C# - программы для ИС „Университеты“ ---

```
using System;
using System.Drawing;
using System.Windows.Forms;

namespace WinListCombo
{
    public partial class Form5 : Form
    {
        int fa = 0;
        string itemU, itemF, itemS;
        public Form5()
        {
            InitializeComponent();
        }

        private void Form5_Load(object sender, EventArgs e)
        {
            comboBox1.Items.Add("Тбилисский Государственный Университет"); // www.tsu.ge
            comboBox1.Items.Add("Грузинский Технический Университет"); // www.gtu.ge
            comboBox1.Items.Add("Тбилисский Гос.Медицинский Университет"); //www.tsmu.edu.ge
        }

        private void button1_Click(object sender, EventArgs e)
        {
            Close();
        }

        private void comboBox1_SelectedIndexChanged(object sender, EventArgs e)
        {
            int selectedIndexU = comboBox1.SelectedIndex;
            Object selectedItemU = comboBox1.SelectedItem;
            itemU = selectedItemU.ToString();
            // MessageBox.Show("SelectedIndexU: " + selectedIndexU.ToString());
            switch (selectedIndexU)
            {
                case 0:
                    comboBox2.Items.Clear();
                    comboBox2.Items.Add("Точных и естественных наук");
                    comboBox2.Items.Add("Юридический");
                    comboBox2.Items.Add("Гуманитарных наук");
                    comboBox2.Items.Add("Социальных и политических наук");
                    pictureBox1.Image = Image.FromFile("D:\\VS_Ru\\WinCombo_IMG\\tsu.jpg");
                    label4.Text = "Тбилиси, Проспект Ильи Чавчавадзе 1";
                    label5.Text = "222-22-22";
                    label6.Text = "www.stu.ge";
                    fa = 100;
                    break;
                case 1:
                    comboBox2.Items.Clear();
                    comboBox2.Items.Add("Архитектуры и строительства");
                    comboBox2.Items.Add("Энергетики");
            }
        }
    }
}
```

```

        comboBox2.Items.Add("Информатики и систем управления");
        comboBox2.Items.Add("Механико-транспортный");
        comboBox2.Items.Add("Бизнес-инженерии");
        comboBox2.Items.Add("Горно-геологический");
        pictureBox1.Image = Image.FromFile("D:\\VS_Ru\\WinCombo_IMG\\gtu.jpg");
        label4.Text = "Тбилиси, ул. М. Костава, 77";
        label5.Text = "237-37-37";
        label6.Text = "www.gtu.ge";
        fa = 101;
        break;
    case 2:
        comboBox2.Items.Clear();
        comboBox2.Items.Add("Лечебный (Медицинский)");
        comboBox2.Items.Add("Стоматологический");
        comboBox2.Items.Add("Фармацевтический");
        comboBox2.Items.Add("Общественного здравоохранения");
        comboBox2.Items.Add("спортивной медицины и реабилитации");
        pictureBox1.Image = Image.FromFile("D:\\VS_Ru\\WinCombo_IMG\\meduni.jpg");
        label4.Text = "Тбилиси, Важа-Пшавела 41";
        label5.Text = "239-39-39";
        label6.Text = "www.tsmu.edu.ge";
        fa = 102;
        break;
    case 3:
        comboBox2.Items.Clear();
        comboBox2.Items.Add("Бизнеса ");
        comboBox2.Items.Add("Инженерный ");
        comboBox2.Items.Add("Права ");
        comboBox2.Items.Add("Науки и искусства ");
        pictureBox1.Image = Image.FromFile("D:\\VS_Ru\\WinCombo_IMG\\iliauni.jpg");
        label4.Text = "Тбилиси, Проспект Ильи Чавчавадзе 101";
        label5.Text = "239-39-39";
        label6.Text = "www.iliauni.edu.ge";
        fa = 103;
        break;
    default: break;
}
}

private void comboBox2_SelectedIndexChanged(object sender, EventArgs e)
{
    int selectedIndexF = comboBox2.SelectedIndex;
    Object selectedItemF = comboBox2.SelectedItem;
    itemF = selectedItemF.ToString();
    // MessageBox.Show("SelectedIndexF: " + selectedIndexF.ToString());
    switch (fa)
    {
        case 100: // ТГУ
            switch (selectedIndexF)
            {
                case 0:
                    comboBox3.Items.Clear();
                    comboBox3.Items.Add("ФИЗ - МАТ");
                    comboBox3.Items.Add("Биологии");
                    comboBox3.Items.Add("Химии");
                    comboBox3.Items.Add("Компьютерныч Наук");
                    break;
                case 1:

```

```

        comboBox3.Items.Clear();
        comboBox3.Items.Add("Уголовного права");
        comboBox3.Items.Add("Гражданского права");
        comboBox3.Items.Add("Административного права");
        break;
    case 2:
        comboBox3.Items.Clear();
        comboBox3.Items.Add("Истории Грузии ");
        comboBox3.Items.Add("Географии ");
        comboBox3.Items.Add("Филологии и журналистики ");
        break;
    case 3:
    default: break;
}
break;
case 101: // ГТУ
switch (selectedIndexF)
{
    case 0:
        comboBox3.Items.Clear();
        comboBox3.Items.Add("Гражданского строительства");
        comboBox3.Items.Add("Урбанистики ");
        comboBox3.Items.Add("архитектуры ");
        comboBox3.Items.Add("Железобетонные конструкции");
        break;
    case 1:
        comboBox3.Items.Clear();
        comboBox3.Items.Add("Гидроэнергетики ");
        comboBox3.Items.Add("теплоэнергетики");
        comboBox3.Items.Add("Атомная энергетика");
        break;
    case 2:
        comboBox3.Items.Clear();
        comboBox3.Items.Add("Компьютерной инженерии ");
        comboBox3.Items.Add("Программной инженерии ");
        comboBox3.Items.Add("Искусственный интеллект ");
        break;
    case 3:
    default: break;
}
break;
case 102: // სამედიცინო უნივერსიტეტი
switch (selectedIndexF)
{
    case 0:
        comboBox3.Items.Clear();
        comboBox3.Items.Add("Психиатрии ");
        comboBox3.Items.Add("Хирургии ");
        comboBox3.Items.Add("Кардиологии ");
        comboBox3.Items.Add("Терапии ");
        break;

```



```

        case 1:
            comboBox3.Items.Clear();
            comboBox3.Items.Add("Челюстно-лицевой хирургии ");
            comboBox3.Items.Add("Терапии");
            comboBox3.Items.Add("Протезирования ");
            break;
        case 2:
            comboBox3.Items.Clear();
            comboBox3.Items.Add("Фармацевтическая ");
            comboBox3.Items.Add("Провизорское дело ");
            comboBox3.Items.Add("Диагностическая деятельность ");
            break;
        case 3:
        default: break;
    }
    break;
}
}

private void comboBox3_SelectedIndexChanged(object sender, EventArgs e)
{
    int selectedIndexS = comboBox3.SelectedIndex;
    Object selectedItemS = comboBox3.SelectedItem;
    itemS = selectedItemS.ToString();
    label7.Text = "Вы выбрали: \n-----\n" +
        itemU + "\n" +
        itemF + "\n" +
        itemS ;
}
}
}

```

Рисунки 14.3-14.7 иллюстрируют работу этих приложений в соответствии с различными запросами, которые реализуются практически манипулированием содержаний comboBox-ов.

Рис.14.3. Шаг-1: выбор наименование Университета

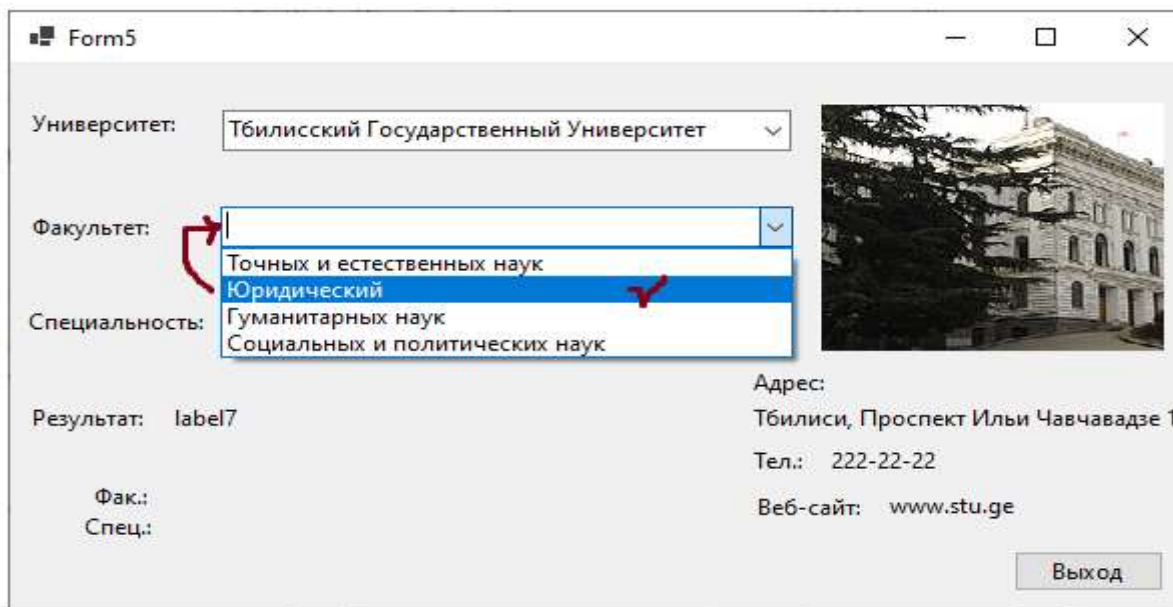


Рис.14.4. Шаг-2: выбор наименования факультета

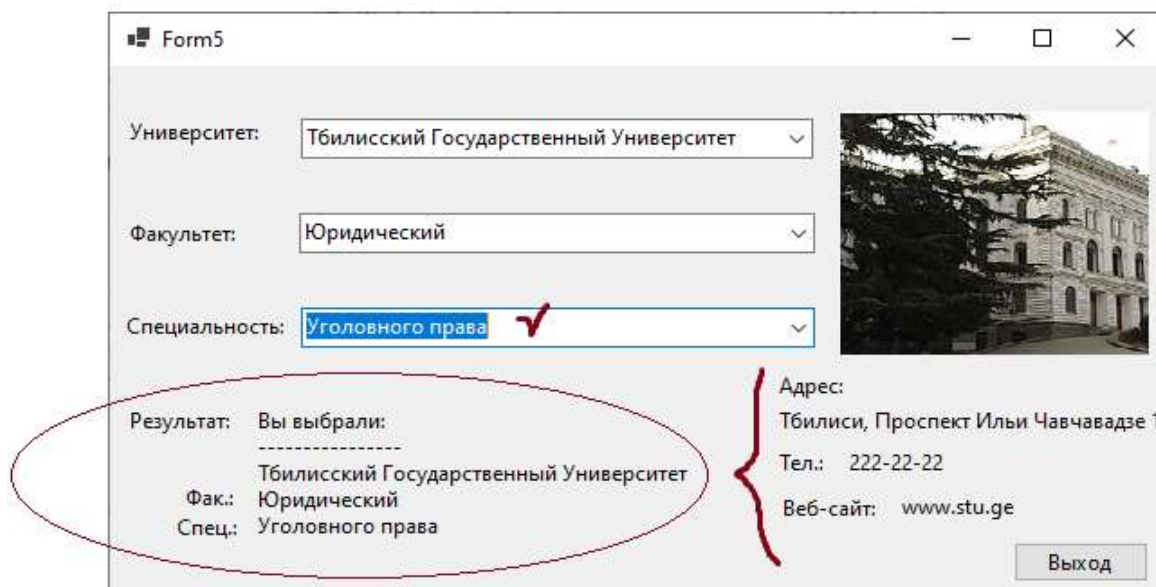


Рис.14.5. Шаг-3: выбор наименования специальности для данного факультета и формирование результата

С помощью трехуровневой реализации системы ComboVox-ов возможно гибко выполнять разные запросы пользователя. При изменении содержания comboVox1, например, на Грузинский Технический Университет, оперативно происходит перестройка содержаний для comboVox2 и comboVox3.

Приведенный листинг C# кода детально описывает последовательные процессы выборочных действий пользователя системы.

На рис. 14.6 и 14.7 иллюстрируется запрос для получения информации о Техническом Университете, его факультетах и по специальностям каждого факультета.

Form5

Университет: Грузинский Технический Университет ✓

Факультет: Энергетики ✓

Специальность: Гидроэнергетики ✓

Результат: Вы выбрали:  
-----  
Грузинский Технический Университет  
Фак.: Энергетики  
Спец.: Гидроэнергетики

Адрес:  
Тбилиси, ул. М. Костава, 77  
Тел.: 237-37-37  
Веб-сайт: www.gtu.ge

Выход

Рис.14.6. Пример Фак-Энергетики и Спец.-Гидроэнергетики

Form5

Университет: Грузинский Технический Университет

Факультет: Информатики и систем управления ✓

Специальность: Программной инженерии ✓

Результат: Вы выбрали:  
-----  
Грузинский Технический Университет  
Фак.: Информатики и систем управления  
Спец.: Программной инженерии

Адрес:  
Тбилиси, ул. М. Костава, 77  
Тел.: 237-37-37  
Веб-сайт: www.gtu.ge

Выход

Рис.14.7. Пример Фак-Информатики и Спец.-Программной инженерии

Возможно расширение системы с помощью реальных данных, хотя для реализации приложения существуют другие способы и методы (например, с применением баз данных, куда помещается информация большого объема), которые существенно сократят объем C#-кода. Эти вопросы рассмотрим в дальнейшем.

➤ **Самостоятельная работа:**

1. Постройте форму “Размен валюты” для выбора иностранной валюты одним ComboBox-ом. Двумя TextBox-ами для ввода количества Лари и значения текущего Курса обмена иностранной валюты;

2. Создайте проект WinForms с кодом C#: CV студента. Используйте визуальные компоненты: 4 панели на форме - [личные данные: личный N, фамилия, имя, дата рождения, пол], вуз: [название вуза, факультет, группа]; Языки: [русский, грузинский, английский и др. /с CheckBox/], адрес: [город, улица-N, e-mail].

## Глава 5

### Визуальные компоненты программирования диалоговых процедур

#### Лабораторная работа N15

##### 5.1. Стандартные диалоговые средства языка C#

В языке визуального программирования C# существует пять видов диалогового класса, которые часто применяются в процессе проектирования программных проектов: OpenFileDialog, SaveFileDialog, FolderBrowserDialog, ColorDialog и FontDialog. Рассмотрим эти диалоги более детально (Рис.2.18).

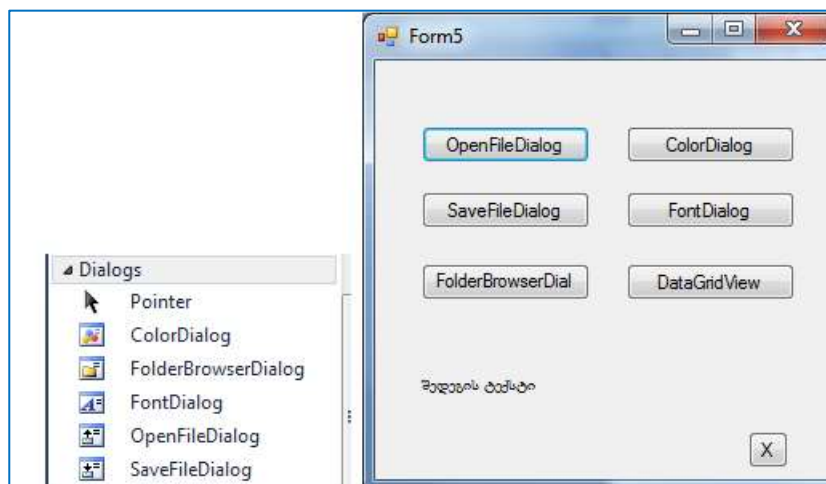


Рис.15.1.

##### 5.1.1. OpenFileDialog

Назначением объекта класса OpenFileDialog является выбор файла согласно указанного фолдера (InitialDirectory), фильтра (Filter), типа файла и заголовка диалогового поля (Title) (Рис.15.2 и 15.3). Результат диалога для свойства имени файла будет FileName.

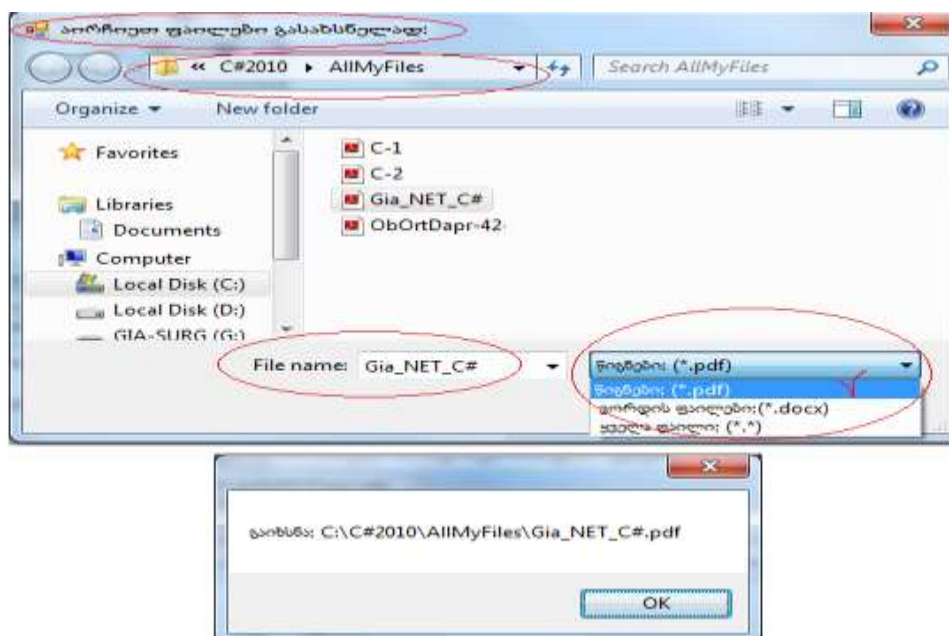


Рис.15.2. Положительный результат

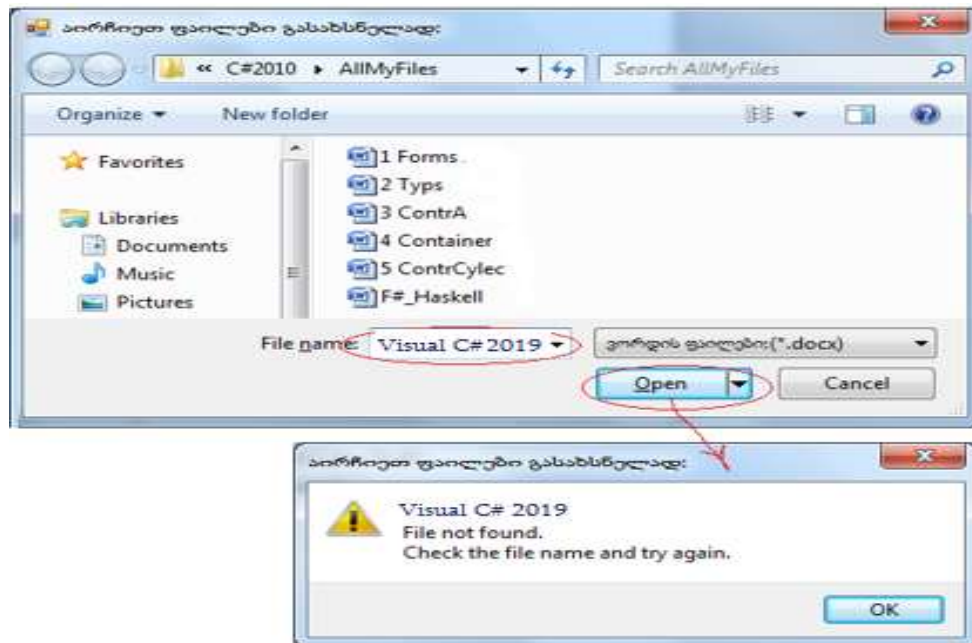


Рис.15.3. Безрезультатное окончание

```
// --- Листинг --- OpenFileDialog -----
private void button1_Click(object sender, EventArgs e) // fileOpenDialog
{
    OpenFileDialog f = new OpenFileDialog();
    f.InitialDirectory = "c:\\C#2010\\AllMyFiles";
    f.Filter = "книги: (*.pdf)|*.pdf| " + " файлы ворда:(*.docx)|*.docx| " + " все файлы: (*.*)|*.*";
    f.Title = "выберите файлы для открытия:";
    if (f.ShowDialog() == DialogResult.OK)
        MessageBox.Show("открылся: " + f.FileName);
    else
        MessageBox.Show("окончание без результата !");
}

```

### 5.1.2. SaveFileDialog

Назначением объекта класса SaveFileDialog является выбор или ввод того файла, который надо хранить. Для осуществления хранения или диалога надо указать фолдер (InitialDirectory), фильтр (Filter), тип файла и заголовок диалогового поля (Title).

```
// листинг --- SaveFileDialog -----
private void button2_Click(object sender, EventArgs e)
{
    SaveFileDialog fs = new SaveFileDialog();
    fs.InitialDirectory = "c:\\C#2010\\AllMyFiles";
    fs.Filter = "книги: (*.pdf)|*.pdf| " + " файлы ворда:(*.docx)|*.docx| " + " все файлы: (*.*)|*.*";
    fs.Title = "выбор файлов для хранения:";
    if (fs.ShowDialog() == DialogResult.OK)
        MessageBox.Show("хранение: " + fs.FileName);
    else
        MessageBox.Show("окончание без результата !");
}

```

### 5.1.3. FolderBrowserDialog

Назначением объекта класса FolderBrowserDialog является выбор каталога (фолдера), который будет для последующих программных процедур базовой точкой. Возможно также создание нового каталога. Перед открытием диалоговой формы необходимо задание следующих параметров: RootFolder: каталог, который должен появиться в диалоговом поле, ShowNewFolderButton: указание необходимого бутона для создания нового каталога, Description: заголовок диалогового поля.

// листинг --- FolderBrowserDialog -----

```
private void button3_Click(object sender, EventArgs e)
{
    FolderBrowserDialog fb = new FolderBrowserDialog();
    fb.RootFolder = Environment.SpecialFolder.MyDocuments;
    fb.ShowNewFolderButton = false;
    fb.Description = "выбор каталога";
    if (fb.ShowDialog() == DialogResult.OK)
        MessageBox.Show("доступ к каталогу: " + fb.SelectedPath);
    else
        MessageBox.Show("окончание без результата!");
}
```

Строкой: fb.RootFolder = Environment.SpecialFolder.MyDocuments;

в нашем случае, в качестве каталога выбран “MyDocuments”.

Строкой: fb.ShowNewFolderButton = false; - новый каталог не создается, но если есть “true”, тогда на форме появляется кнопка “Make New Folder” (Рис.15.4).

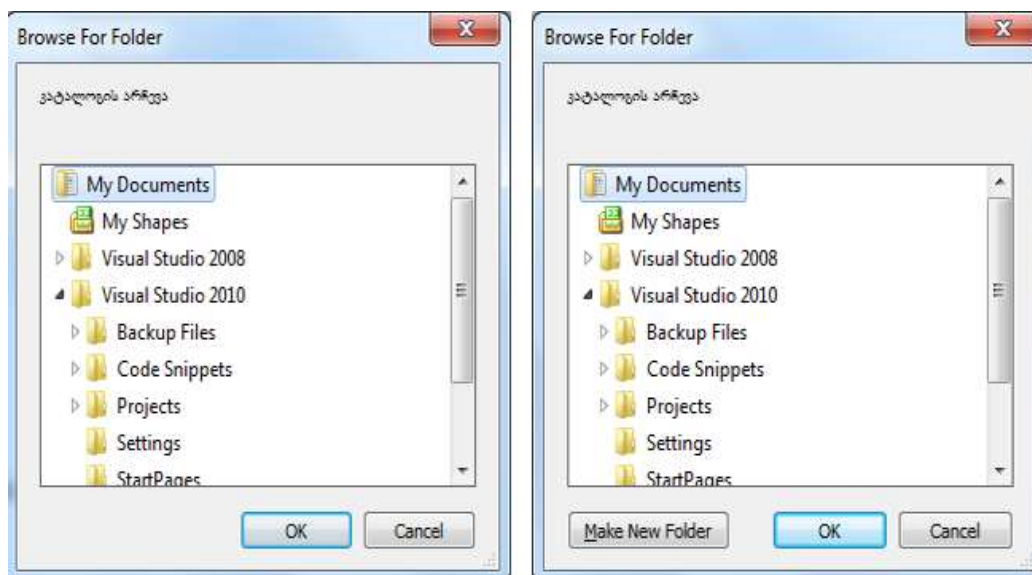


Рис.15.4

#### 5.1.4. ColorDialog

Назначением объекта класса ColorDialog является выбор цвета для отмеченного элемента, который расположен на форме (см. листинг и рис. 15.5).

```
// ლისტინგი --- ColorDialog -----
private void button4_Click(object sender, EventArgs e)
{
    ColorDialog cd = new ColorDialog();
    if (cd.ShowDialog() == DialogResult.OK)
        label1.ForeColor = cd.Color;
    else
        MessageBox.Show("უშედეგოდასასრული");
}

```

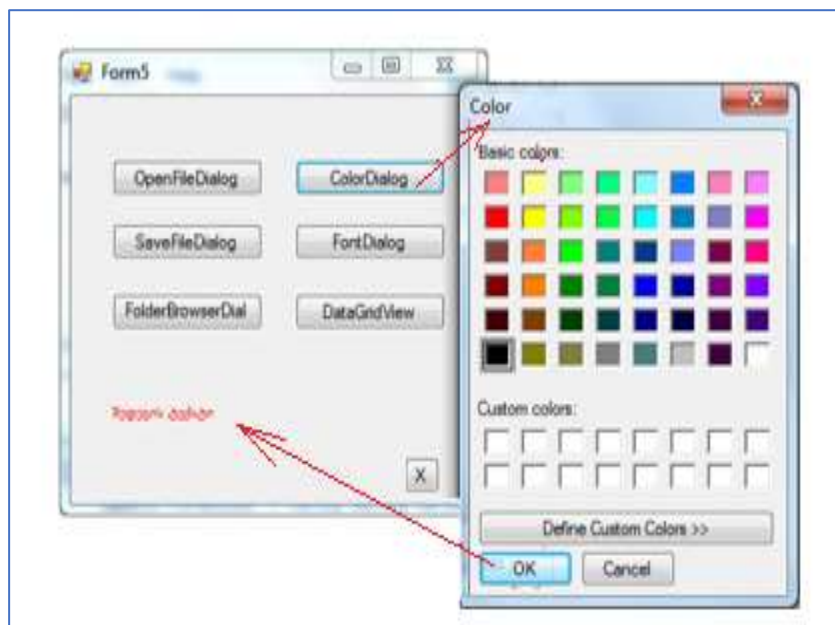


Рис.15.5.

#### 5.1.5. FontDialog

Назначением объекта класса FontDialog является выбор шрифта (фонта) для отмеченного элемента, который расположен на форме (см. листинг и рис. 15.6).

```
// ლისტინგი --- FontDialog -----
private void button5_Click(object sender, EventArgs e)
{
    FontDialog fd = new FontDialog();
    fd.ShowColor = true;
    fd.MaxSize = 22;
    fd.MinSize = 8;
    if (fd.ShowDialog() == DialogResult.OK)
    {

```

```
label1.Font = fd.Font;  
label1.ForeColor = fd.Color;  
}  
else  
    MessageBox.Show("окончание без результата");  
}
```

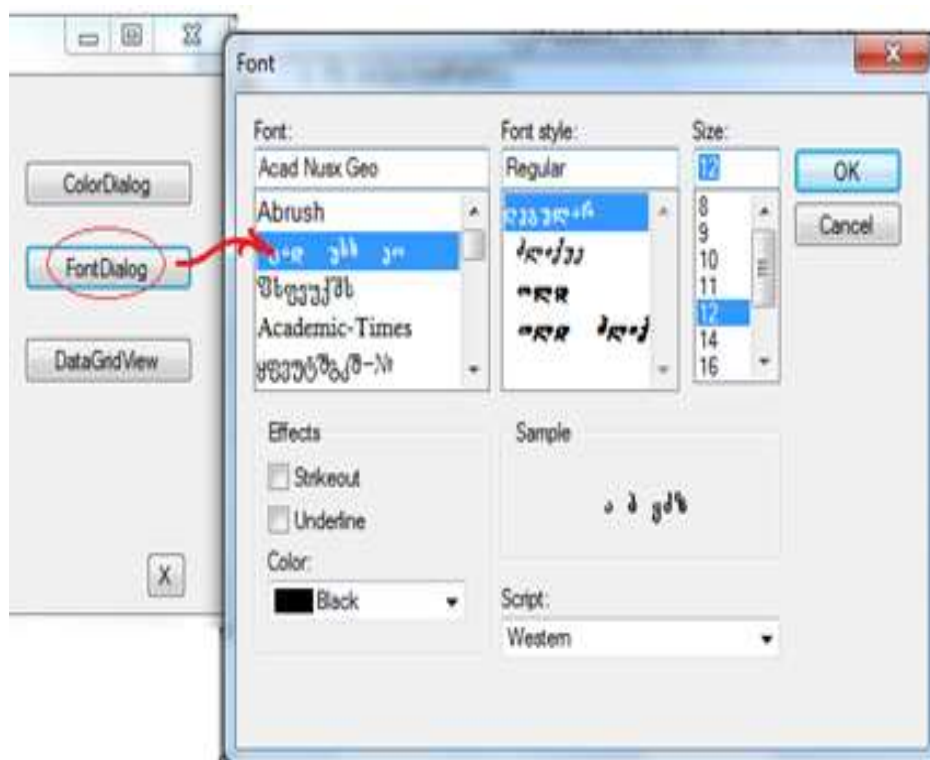


Рис.15.7



## Лабораторная работа N16

### 5.2. Программные средства построения

#### главного меню

**Цель работы:** изучение визуальных элементов управления для построения главного меню компьютерной системы.

**Задача\_1:** на Form1 построим главное меню “Университеты” с подпунктами “ Факультеты” и “Кафедры”. Для создания главного меню из ToolBox-а перенесем на форму элемент MenuStrip. На форме появится изображение, показанное рис.16.1. Вносим пункты меню (рис.16.2). Это осуществляется вертикально и/или горизонтально.

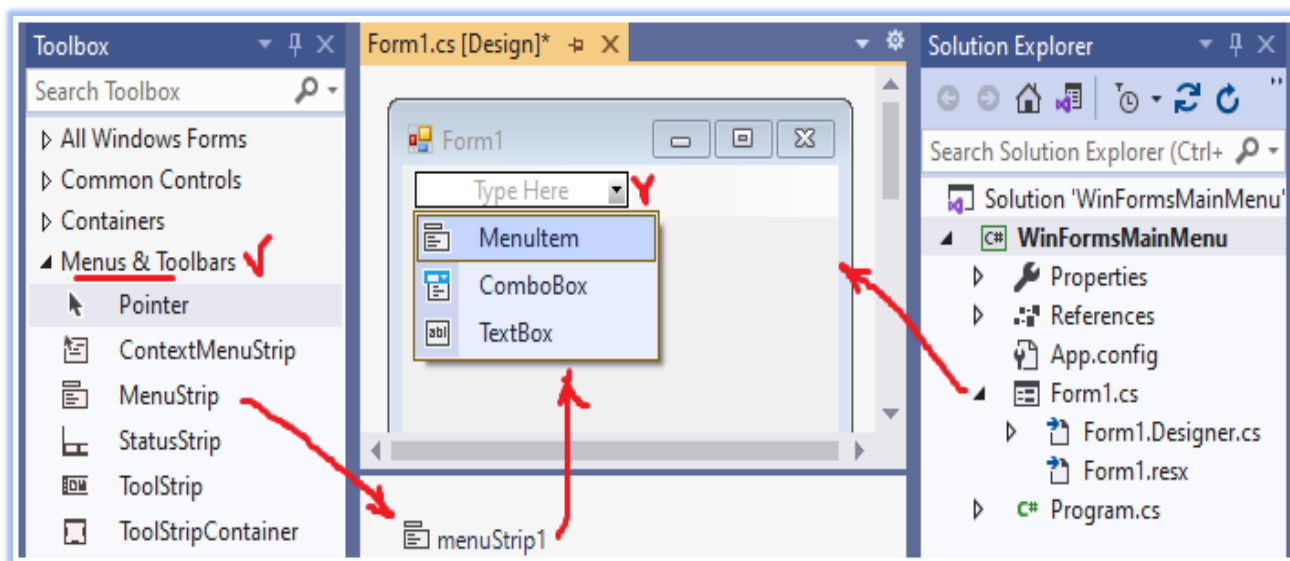


Рис.16.1. Не-визуальный элемент *menuStrip* для создания главного меню интерфейса

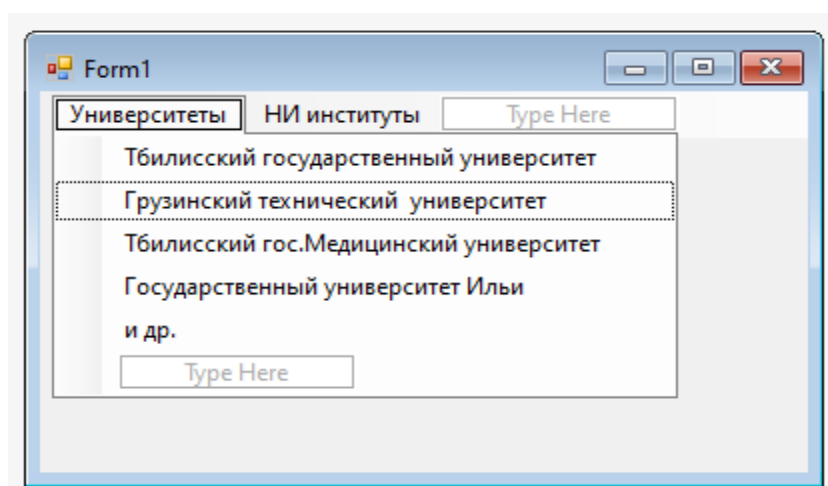


Рис.16.2.

Для каждой строки возможно создание подуровней (рис.16.3).

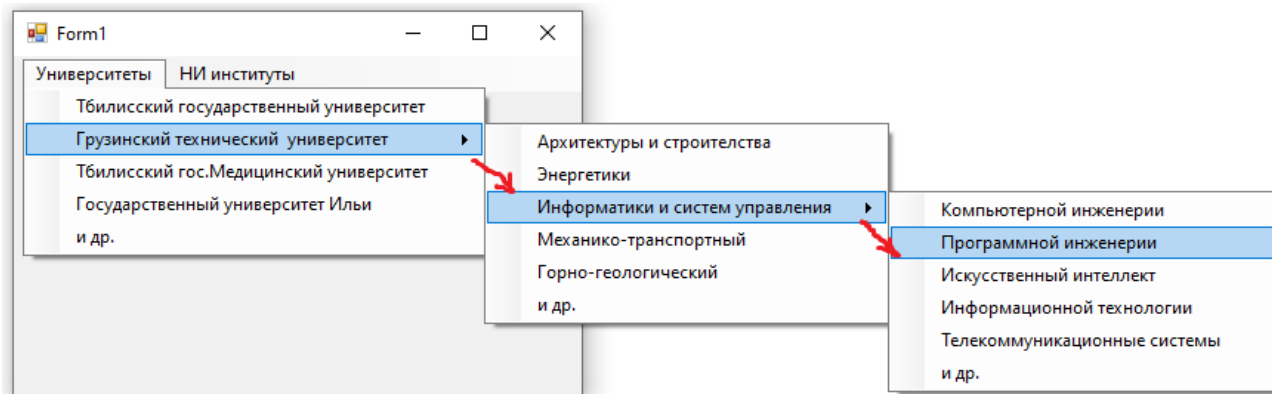


Рис.16.3. Трехуровневое меню

Перемещение пунктов (строчек) меню возможно в режиме Form1[Design] с помощью левой клавиши мыши Drag&Drop (перенос).

Пункты главного меню могут быть трех видов (рис.16.4).

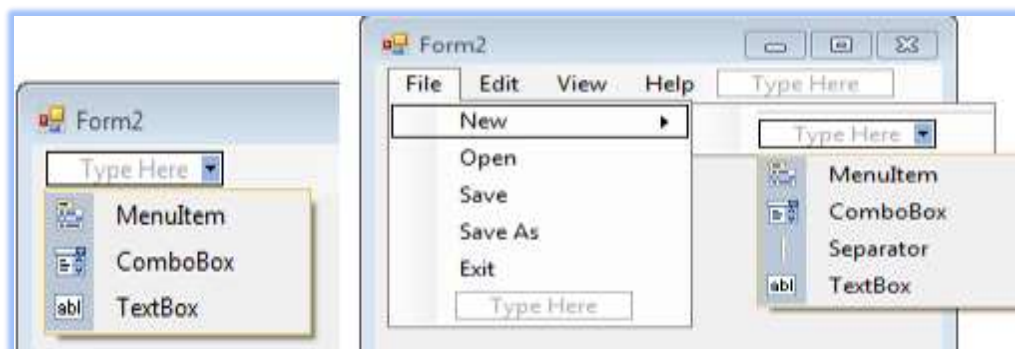


Рис.16.4

- MenuItem – обыкновенная строка меню для ввода;
- ComboBox – из которого осуществляется выбор или ввод;
- TextBox – для ввода текста.

Для подменю возможен пункт Separator. Его назначение – выделение пунктов линией, которая создает для пользователя визуальный комфорт (рис.16.5):

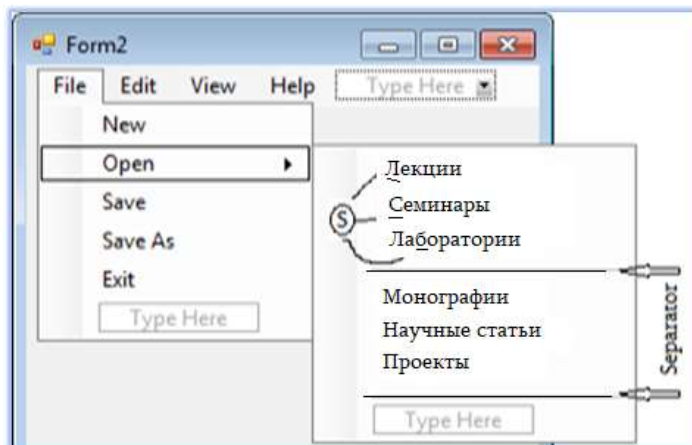


Рис.16.5. Применение Separator-а.

В меню часто применяют *выделение одной буквы строки* (подчеркивание), с помощью которой задействуется этот пункт. На рисунке это показано символом S.

С помощью пунктов меню должно выполниться определенное задание (определение пути доступа к нужной информации и выбор). Поэтому эти пункты связаны с событиями и методами. Главным типом события является Click и к нему привязан код процедуры, которая должна выполниться. Рассмотрим этот вопрос на примере.

**Задача\_2:** на форме (Рис.16.6) построим главное меню, пункт которого “лекции” копирует в textBox1 записанную наименование темы лекции в label1. Выбором пункта Exit программа закончит работу.

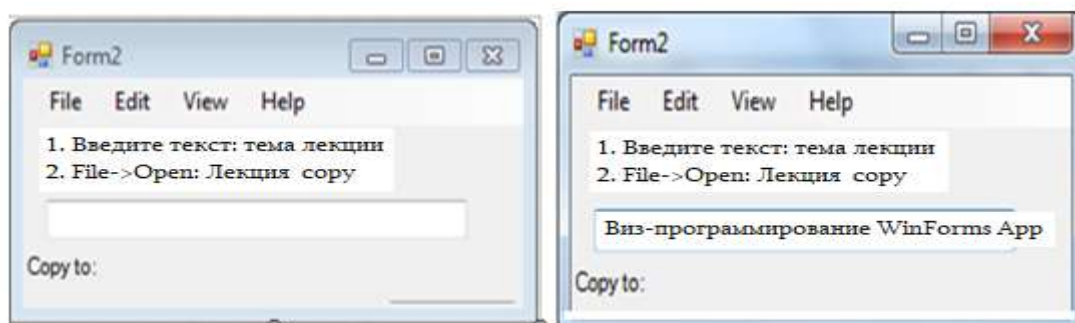


Рис. 16.6. Макет формы

Выбором пункта главного меню “File -> Open -> Лекция” в textBox-е записанная строка “Визуальное программирование WinForms App” копируется в поле label9, которое расположено правее “Copy to” label-а (оно не видно). В листинге показан код обработки этого события:

```
//---Листинг--- Главное меню: File -> Open -> Лекций -----
private void ToolStripMenuItem_Click(object sender, EventArgs e)
{
    label9.BackColor = Color.Red;
    label9.ForeColor = Color.White;
    label9.Text = textBox1.Text;
}
private void exitToolStripMenuItem_Click(object sender, EventArgs e)
{
    Close();
}
```

Результат показан на рис.16.7 с белым текстом и красным фоном.

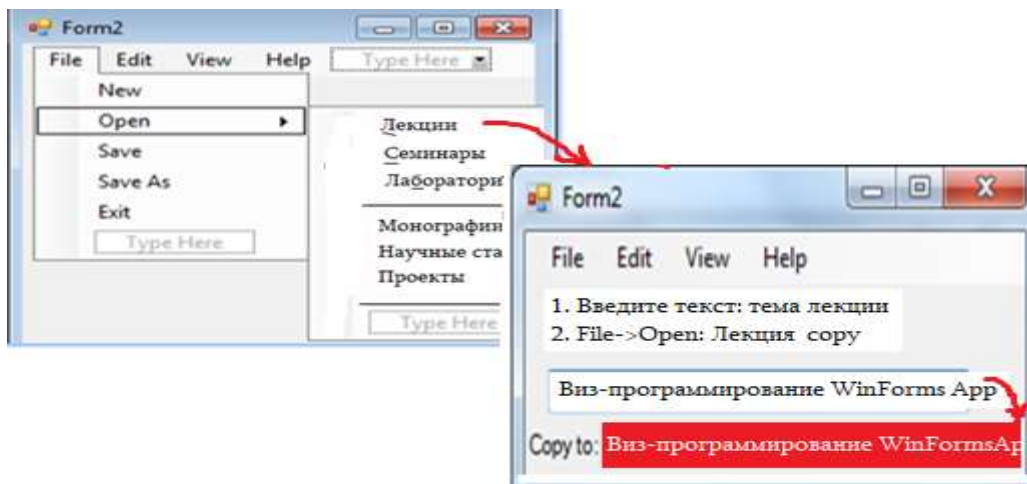


Рис.16.7. Запись строки из textBox1 в label Copy to

## Лабораторная работа N17

### 5.3. Программные средства для построения графического меню

**Цель работы:** изучение процессов построения графического меню и диалоговых процедур с помощью применения визуальных элементов управления.

При построении компьютерных систем в интерфейсе кроме главного меню часто применяют графические пиктограммы (icons), что повышает эффективность и гибкость его применения. В языке C# таким визуальным элементом является toolStrip панели Menus&Toolbars. Его переносом на форму получим картину, показанную на рис.17.1. Необходимо выбрать: Button, Label, ComboBox и т.д.

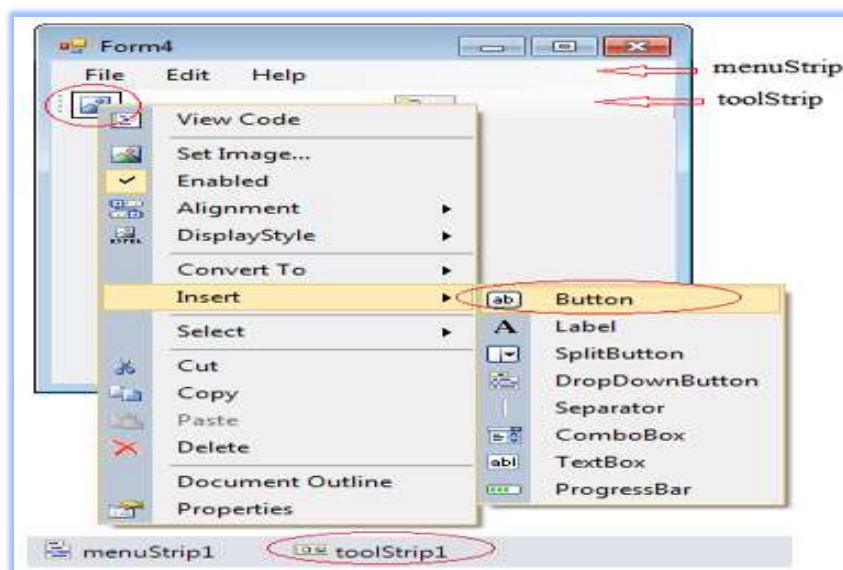


Рис.17.1

Здесь возможна замена стандартно полученной пиктограммы с помощью правой клавиши мыши, сперва выбором свойства Image в Properties-e, а затем Import-ированием необходимой пиктограммы из диалогового окна (рис.17.2).

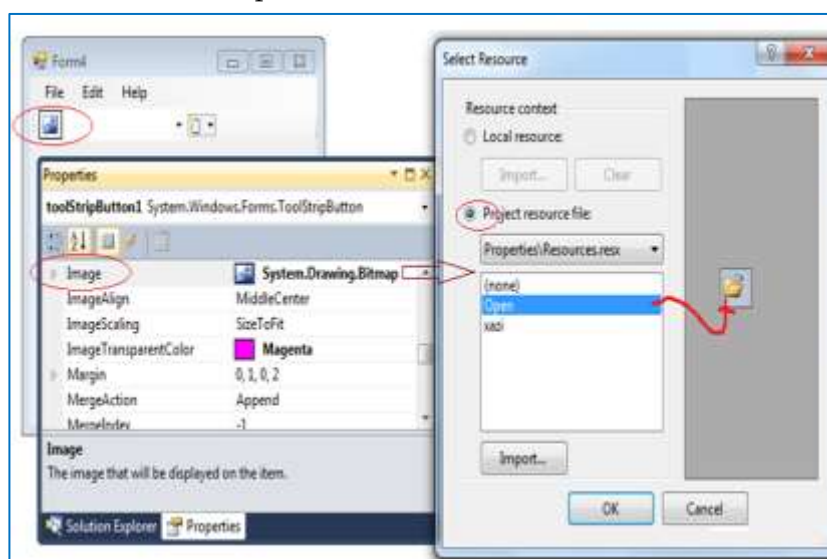


Рис.17.2

В свойство Text новой пиктограммы запишите слово, соответствующее ее функции, например Open. На рис.17.3 виден полученный результат.

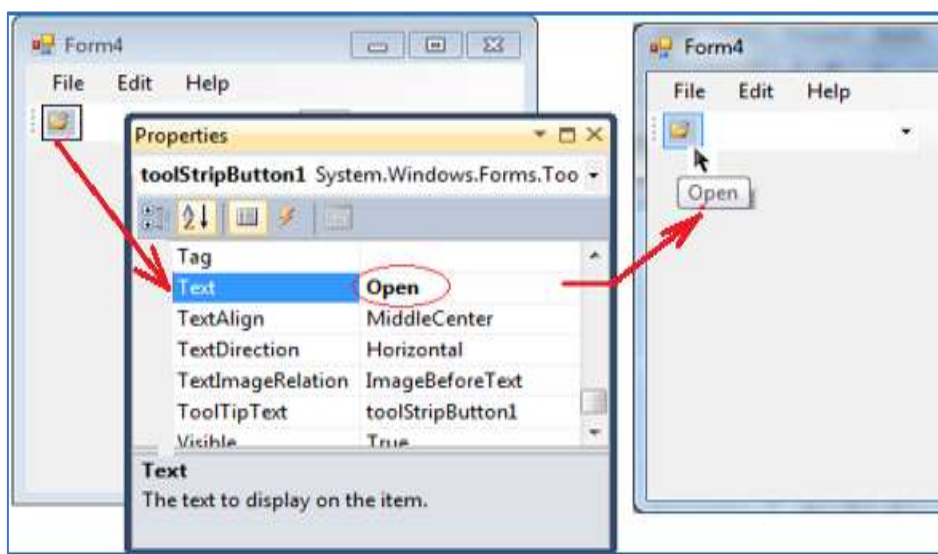


Рис.17.3

Конечные результаты, полученные после записи новых пиктограмм, среди которых одна типа ComboBox, показаны на рис.17.4.



Рис.17.4

### ➤ Самостоятельная работа:

Построить главное меню и соответствующее графическое меню для Windows Forms App: "Факультет": с горизонтальными пунктами - "Кафедры" -> "Группы", "Предметы", "Help"; Также с 3-4 пунктами по вертикали в каждой горизонтали. Например, «Группы: -> выбор Гр\_Ном». После выбора группы на экране появятся Фамилии и Имена студентов в этой группе и их эл-почта.

## Лабораторная работа N18

### 5.4. Программные средства для построения контекстного меню

Цель работы: применение элемента ContextMenuStrip ToolBox-панели. Приведение в соответствии с необходимыми функциями значений строк контекстного меню.

**Задача\_1:** построить форму, например в виде, показанном на рис.18.1, на которой размещены элементы label1 и textBox1. Для этих элементов необходимо создать два контекстных меню. В textBox1 записанная строка с контекстным меню надо скопировать в label1. Затем перенесенная в label1 форма строки (стиль, размер) должна измениться из контекстного меню.



Рис.18.1

На рис.18.2-а,б показаны помещенные в Form4 для элементов label1 и textBox1 созданные контекстные меню ContextMenuStrip1 и ContextMenuStrip2. Теперь их надо “привязать” к элементам textBox1 и label1 Form4-е для того, чтобы работала правая клавиша мыши и при установке курсора на эти элементы появилось соответствующее контекстное меню.

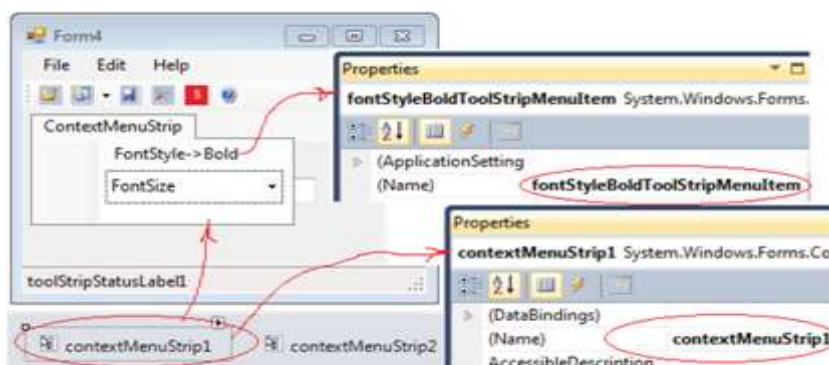


Рис.18.2-а

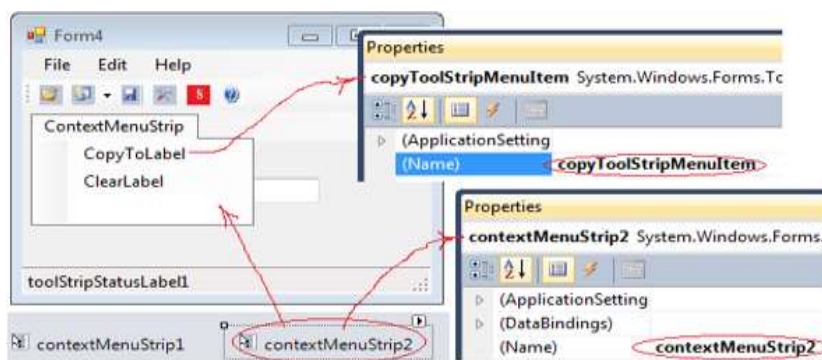


Рис.18.3-б

Для этого в свойство ContextMenuStrip Properties-а label1-го запишем значение ContextMenuStrip1 (Рис.18.4). Также для textBox1 запишем ContextMenuStrip2.

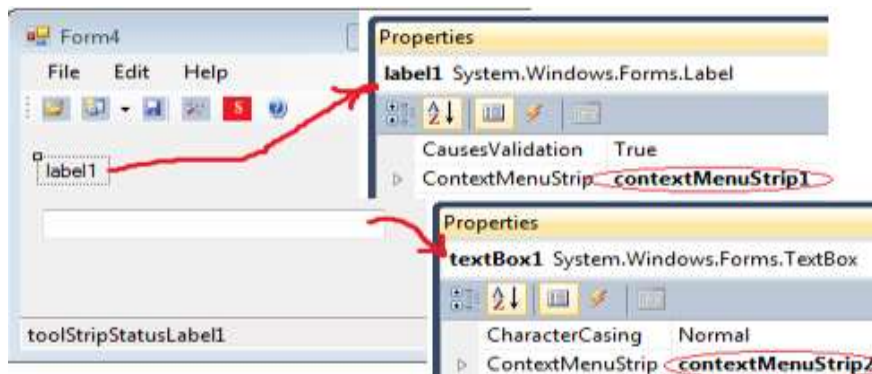


Рис.18.4

Программный код приведен в следующем листинге, а результаты на рисунках 18.5-18.7.

// --- Листинг – изменение шрифта ----

```
using System;
using System.Drawing;
using System.Windows.Forms;
using System.Threading;

namespace WinMenus
{
    public partial class Form4 : Form
    {
        double DamtavrDro;
        public Form4()
        {
            InitializeComponent();
        }
        private void Form4_Load(object sender, EventArgs e)
        {
            toolStripStatusLabel1.Text = DateTime.Today.ToShortDateString();
        }
        private void Stop_Click(object sender, EventArgs e)
        {
            DamtavrDro = 0;
            timer1.Enabled = true;
        }
        private void timer_Tick(object sender, EventArgs e)
        {
            DamtavrDro += 0.1;
            if (DamtavrDro >= 5)
                Close();
            else
                toolStripProgressBar1.Value = (int)DamtavrDro;
                textBox1.Text = DamtavrDro.ToString();
        }

        private void copyToolStripMenuItem_Click(object sender, EventArgs e)
        {
            label1.Text = textBox1.Text;
            if (label1.Text == "")
                label1.Text = "(leer)";
        }
    }
}
```

```

privatevoid clearLabelToolStripMenuItem_Click(object sender, EventArgs e)
{
    label1.Text = "";
}
privatevoid fontStyleBoldToolStripMenuItem_Click(object sender, EventArgs e)
{
    label1.Font = newFont(label1.Font.FontFamily, label1.Font.Size,
        label1.Font.Style ^ FontStyle.Bold);
    fontStyleBoldToolStripMenuItem.Checked = !fontStyleBoldToolStripMenuItem.Checked;
}
privatevoid fontSize16ToolStripMenuItem_Click(object sender, EventArgs e)
{
    label1.Font = newFont(label1.Font.FontFamily, label1.Font.Size,
        label1.Font.Style ^ FontStyle.Italic);
    fontStyleBoldToolStripMenuItem.Checked = !fontStyleBoldToolStripMenuItem.Checked;
}

privatevoid toolStripComboBox1_TextChanged(object sender, EventArgs e)
{
    double FontSize;
    try
    {
        FontSize = Convert.ToDouble(toolStripComboBox1.Text);
    }
    catch
    {
        FontSize = 8.25;
    }

    label1.Font = newFont(label1.Font.FontFamily, (float)FontSize, label1.Font.Style);
}

privatevoid toolStripComboBox1_Click(object sender, EventArgs e)
{
    toolStripComboBox1.Items.Clear();
    toolStripComboBox1.Items.Add("8,25");
    toolStripComboBox1.Items.Add("12");
    toolStripComboBox1.Items.Add("16");
    toolStripComboBox1.Items.Add("22");
    toolStripComboBox1.SelectedIndex = 0;
}
}
}

```

Результаты (рис.18.5).



Рис.18.5



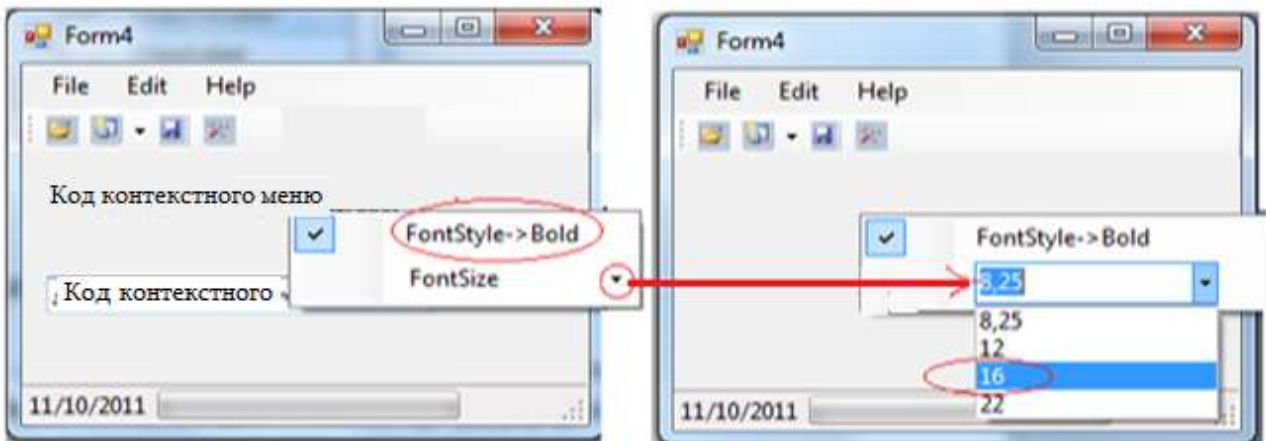


Рис.18.6

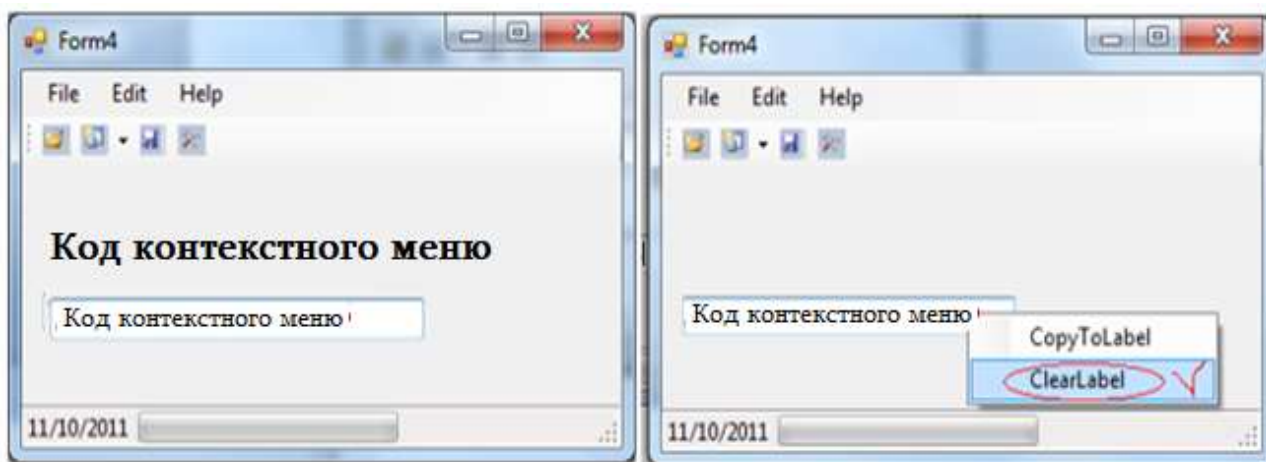


Рис.18.7

➤ **Самостоятельная работа:**

1. Используйте материал из главы 5 для разработки главного и контекстного меню для вашей курсовой работы. Напишите и отладьте их C# - кодс.
2. Построить проект программной аппликации для модели «Светофора», который будет работать как реальный светофор по принципу смены цветов через определенные промежутки времени. На рисунке 18.8 показан пример такой формы.



Рис.18.8

## Глава 6

### Визуальные средства работы с базами данных языка C#

Проектирование приложений на платформе Visual Studio.NET Framework с использованием языка C# подразумевает применение стандартных визуальных элементов доступа к данным (Рис.19.1-а).

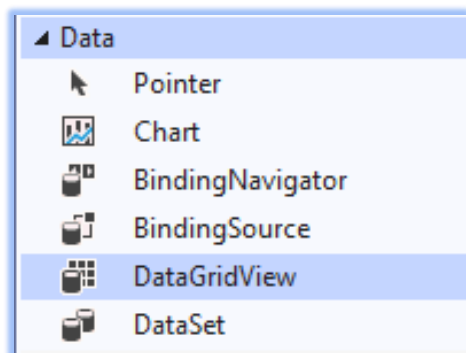
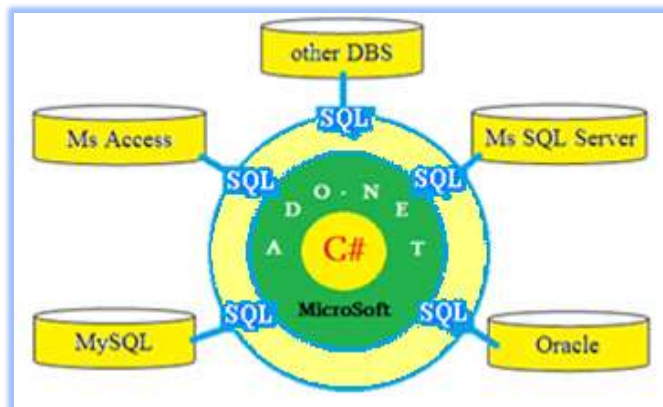


Рис.19.1. Компоненты Toolbox Data



бб.19.2. C# <-> ADO.NET <-> SQL <-> DBS

Для связи программы с базой данных (SQL Server, Access, Oracle и др.) применяется драйвер ADO.NET (ActiveX Data Object) (рис.19.2). ADO.NET используется как для построения Web-сервисов (ADO.NET Data Services), так и в технологиях Майкрософта WPF, WCF и ASP.NET.

Рассмотрим Data элементы Toolbox – а.

BindingNavigator – создание стандартных средств поиска и изменений данных в Windows Form (рис.19.3).

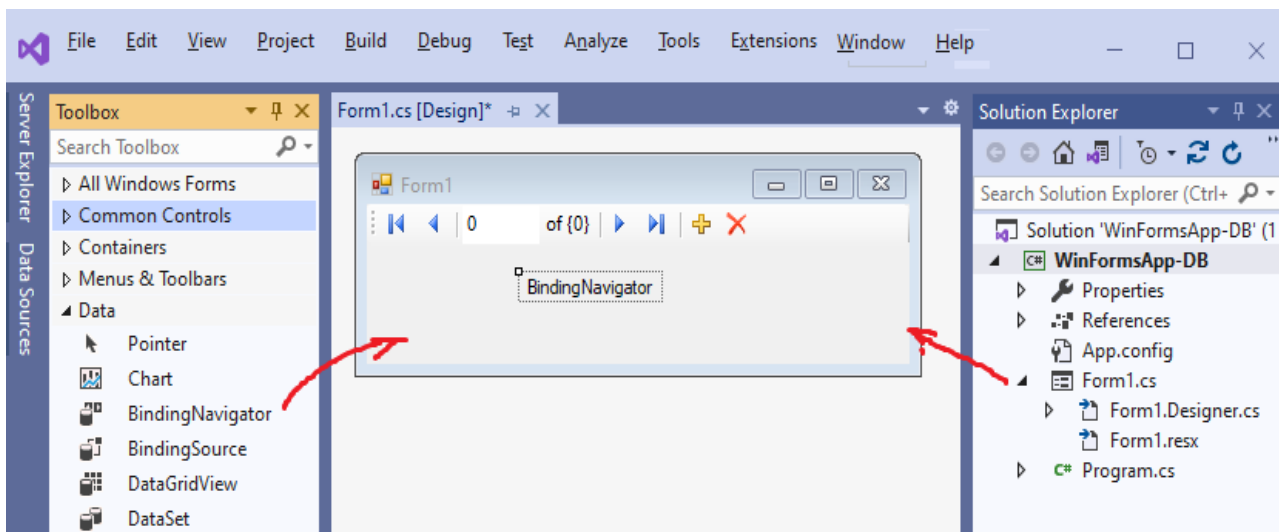


Рис. 19.3

BindingSource – упрощает процесс привязки элементов управления к источнику данных. В дальнейшем все операции, такие как перемещение, упорядочение, фильтрация и обновление осуществляются вызовом компонент BindingSource-а;

DataGridView – редактирование и отображение табличных данных;

DataSet – является основным объектом ADO.NET. Он получает данные из базы и помещает в кеш-память для дальнейшей обработки.

## Лабораторная работа N19

### 6.1. Элемент управления представлением таблиц DataGridView

**Цель работы:** изучение элементов управления работой с таблицами на базе DataGridView.

Для отображения простейших списков и полей данных применяются ListBox и ComboBox. Для представления полей, массивов эффективным средством являются таблицы (Tables), которые состоят из строк и граф. В языке C# для отображения такого объекта применяется элемент управления DataGridView.

На рис.19.4 приведен Windows Forms после переноса из ToolBox-а элемента DataGridView и исходная ситуация.

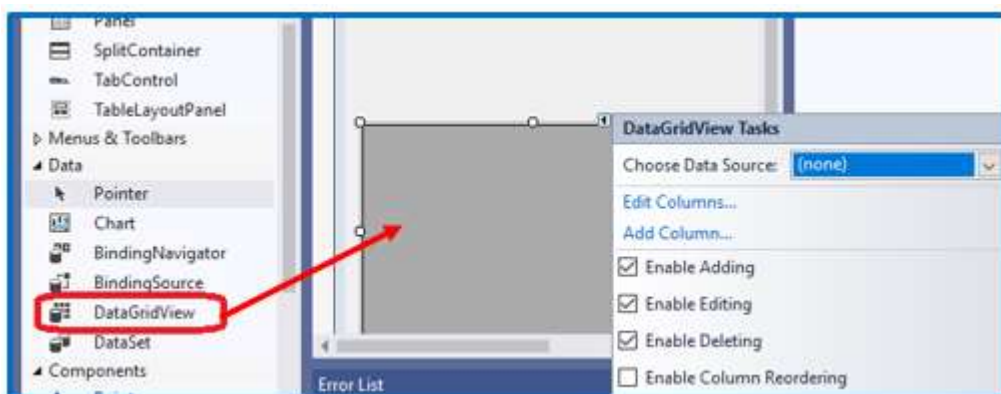


Рис.19.4

В диалоговом режиме для внесения полей (столбцов) таблицы выбираем Add Columns и переходим в окно, показанное на рис.19.5. Занесем последовательно атрибуты “Студенты”, например No, First\_Name (имя), Last Name (фамилия), Birth\_data (дата рождения) и т.д.

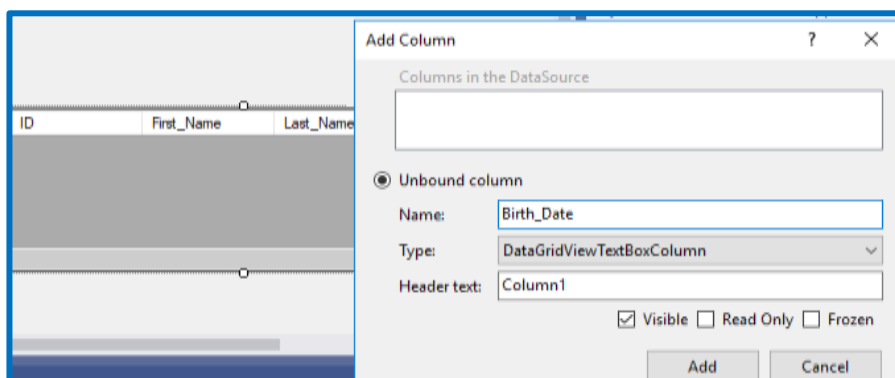


Рис.19.5

После работы программы получим таблицу, приведенную на рис. 19.6.

	No	Имя	Фамилия	Дата рождения
*				

Рис.19.6

Для корректировки полей применяем пункт Edit Columns. Добавим поле Sex (пол) или исправим уже занесенные наименования. Например, на рис.19.6. хотим поле pol заменить русским шрифтом и в тоже время перенести его после поля “имя”. На рис.19.7 и 19.8 показан этот случай.

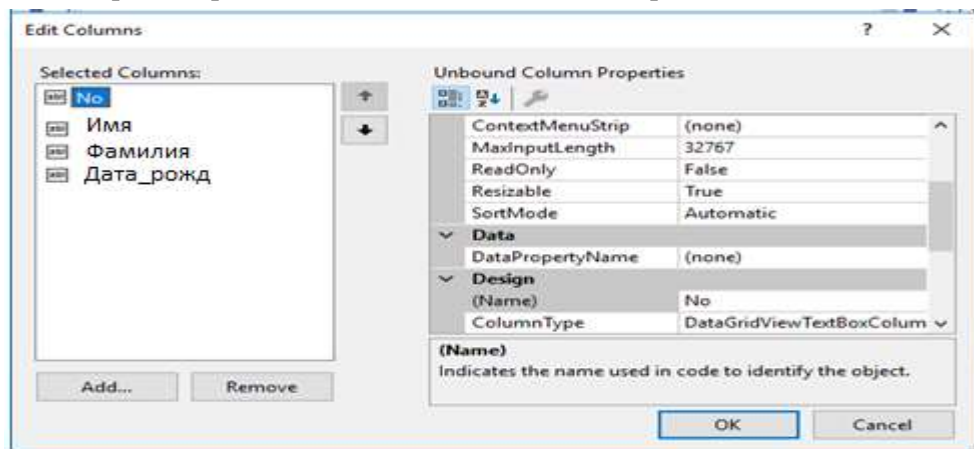


Рис.19.7

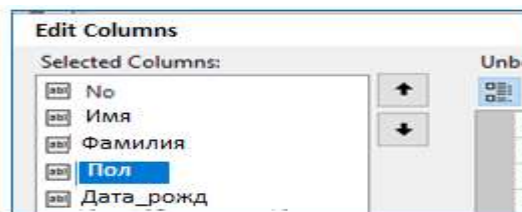


Рис.19.8

Результат изменения показан на рис.19.9.

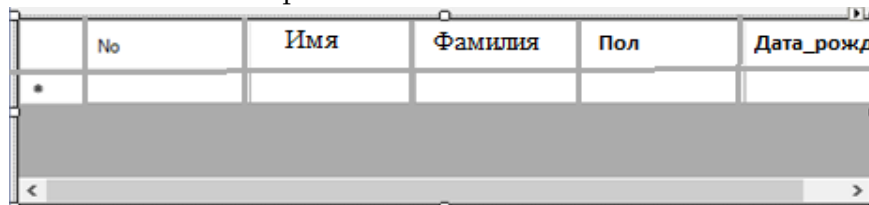


Рис.19.9

Здесь же можно произвести занесение строк данных (Rows). Пример показан на рис.19.10.



Рис.19.10

После окончания работы программы и ее загрузку заново введенные данные теряются, т.е. они не сохранены в памяти. Если хотим, чтобы при запуске проекта данные загрузились бы программно, надо либо применить связь с базой данных (см. следующую лаб.-20), либо в код метода Form2\_Load записать следующие строки (см. листинг):

```
// листинг --- DataGridView -----
private void Form2_Load(object sender, EventArgs e)
{
    int i;
    //---- заполнение полей (столбцов) ----
    dataGridView1.Columns.Add("No", "No");
    dataGridView1.Columns.Add("FirstName", "Имя");
    dataGridView1.Columns.Add("LastName", "Фамилия");
    dataGridView1.Columns.Add("Sex", "Пол");
    dataGridView1.Columns.Add("Dab_Celi", "Год_рожд.");

    //---- установка ширины полей ----
    for (i = 0; i < dataGridView1.Columns.Count; i++)
        dataGridView1.Columns[i].Width = 75;

    //---- заполнение строк -----
    dataGridView1.Rows.Add("1", "Георгий", "Аваков", "М", "2002.01.25");
    dataGridView1.Rows.Add("2", "Александра", "Алиева", "Ж", "2003.05.15");
    dataGridView1.Rows.Add("3", "Иван", "Иванов", "М", "2002.12.20");
    dataGridView1.Rows.Add("4", "Константин", "Петров", "М", "1999.07.31");
    dataGridView1.Rows.Add("5", "Алла", "Пугачева", "Ж", "2003.02.29");
}
```

В результате работы программы получим результат, показанный на рис.19.11.

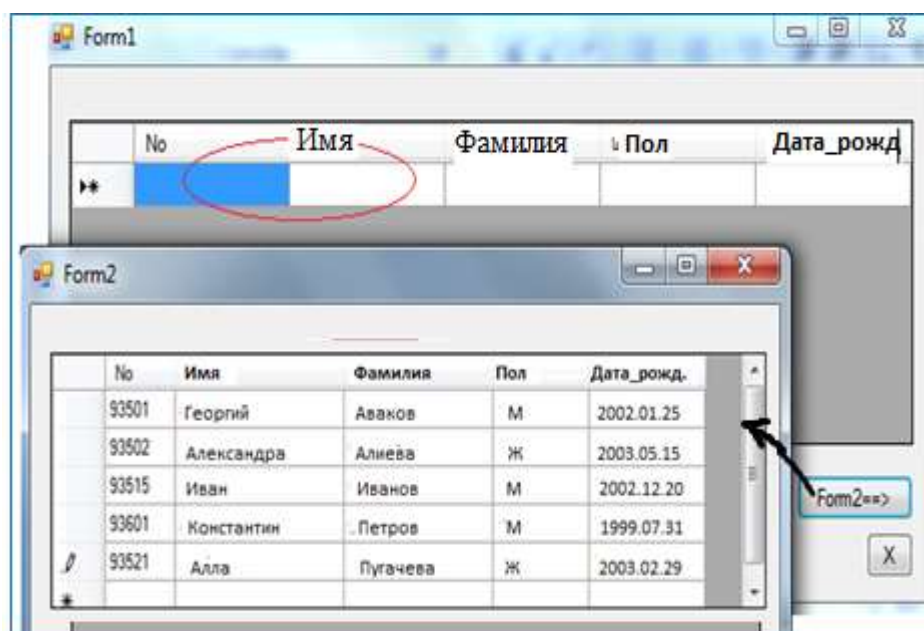


Рис.19.11

Как было сказано, в программный код жестко занесены конкретные данные о студентах. Любое дополнение или изменение требует изменение программы, *что не рекомендуется*. Это можно избежать применением базы данных. Из рис.19.11 видно, что Form1 пустая, а Form2 заполнена исходными данными.

Рассмотрим на примере кода возможности программной обработки внесенных в таблицу DataGridView данных по запросу пользователя.

**Задача\_1:** найти имена и фамилии всех студентов мужского пола, рожденных в 2002 году. Формальная сторона запроса состоит в распечатке полей “фамилия” для значений поля “пол”. Необходимо на Form2 поместить кнопку и привязать к нему программный код, приведенный в следующем листинге:

```
//---Листинг - Выбор в DataGridView строк с условием -----
private void button1_Click(object sender, EventArgs e)
{
    label1.Text = " ";
    label2.Text = "имя и фамилия: все студенты мужского пола рожденные в 2002 году:";
    for (int i = 0; i < dataGridView1.Rows.Count; i++)
    {
        if (dataGridView1.Rows[i].Cells[3].Value == "М" &&
            dataGridView1.Rows[i].Cells[4].Value == "2002")
        {
            label1.Text += dataGridView1.Rows[i].Cells[2].Value + " " +
                dataGridView1.Rows[i].Cells[1].Value + "; ";
        }
    }
}
```

Результат показан на рис.19.12.

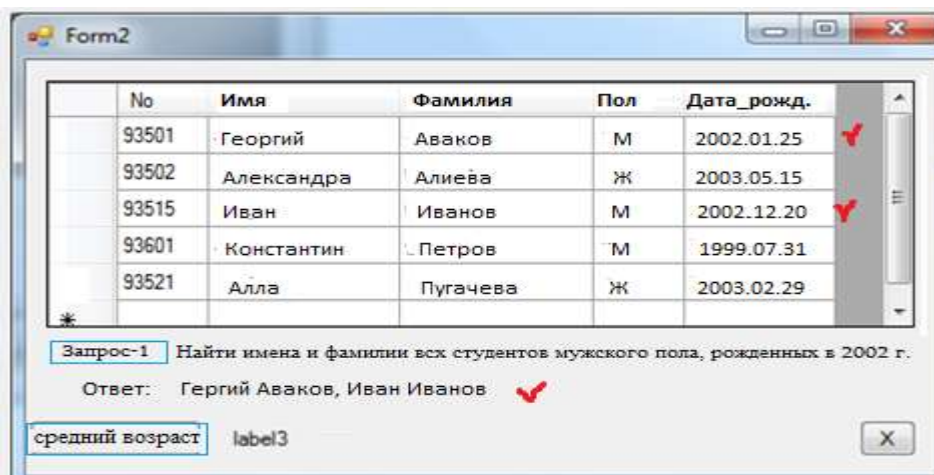


Рис.19.12

**Задача\_2:** составить код для клавиши “средний возраст”, который в label3 выдаст средний возраст всех студентов (рис.19.13).



Рис.19.13. Форма с новой клавишей “средний возраст”.

Программный код приведен в следующем листинге:

```
// листинг --- DataGridView определение среднего возраста -----
private void button2_Click(object sender, EventArgs e)
{
    int Birth_Year, Averag_Age, Sum=0;
    DateTime now = DateTime.Now; // текущая дата - системная
    int age,a,b;
    for (int i = 0; i < dataGridView1.Rows.Count; i++)
    {
        Birth_Year=Convert.ToInt32(dataGridView1.Rows[i].Cells[4].Value);
        a = now.Year;
        b = Birth_Year;
        age = a - b;
        Sum += age;
    }
    Averag_Age = Sum / dataGridView1.Rows.Count;
    label3.Text = Averag_Age.ToString();
}
}
```

Результат показан на рис.19.14.

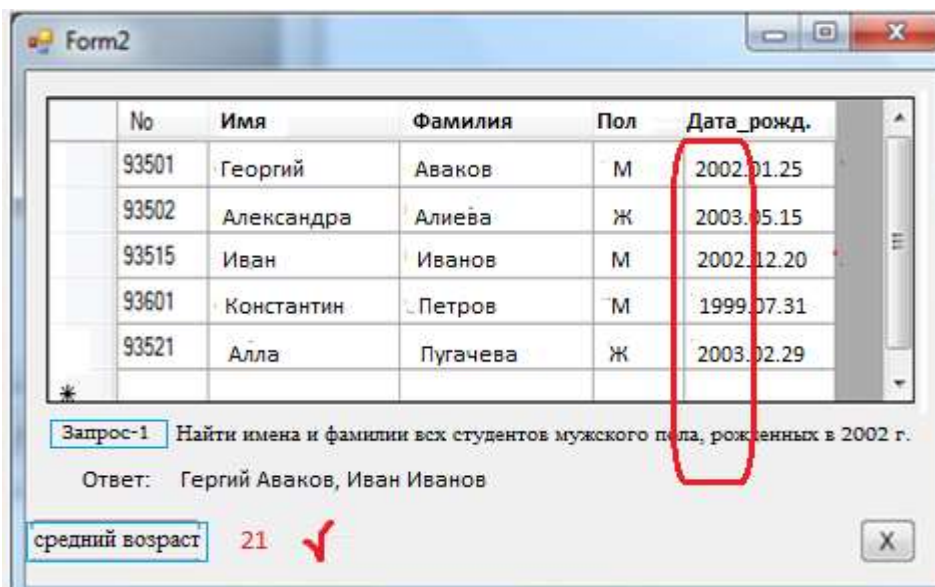


Рис.19.14. Результат запроса „средний возраст всех мужчин“

**Задача 3:** создадим код, который будет работать с таблицей. В частности, установкой курсора мыши на какую-либо ячейку таблицы и кликом, на экран выносятся координаты строки и столбца этой ячейки и записанное в ней данное. Создание события показано на рис.19.15.

```
// листинг --- координаты таблицы -----
private void dataGridView1_CellClick(object sender, DataGridViewCellEventArgs e)
{
    label8.Text = "";
    label6.Text = "Row="+e.RowIndex.ToString();
    label7.Text = "Column="+e.ColumnIndex.ToString();

    if(e.RowIndex>=0 && e.ColumnIndex >=0)
        label8.Text += dataGridView1.Rows[e.RowIndex].Cells[e.ColumnIndex].Value;
}
```

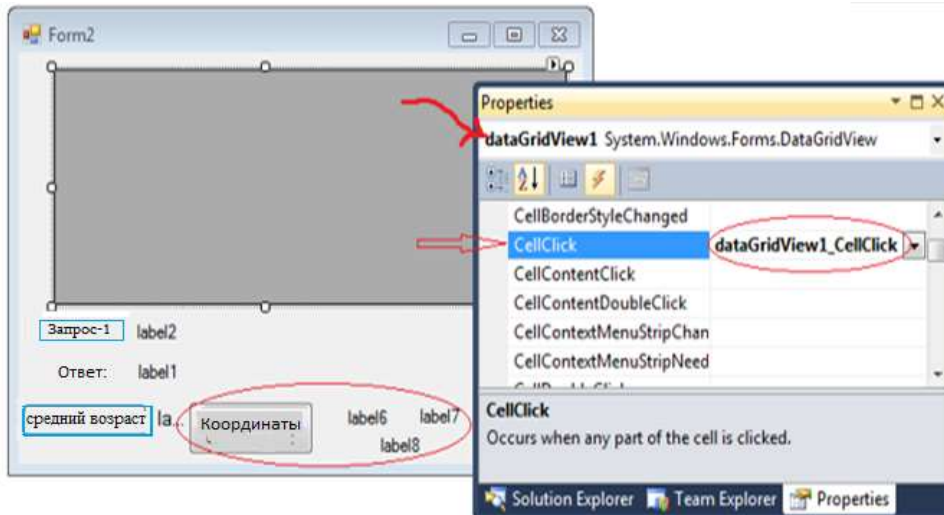


Рис.19.15

Результаты приведены на рис.19.16 а,б.

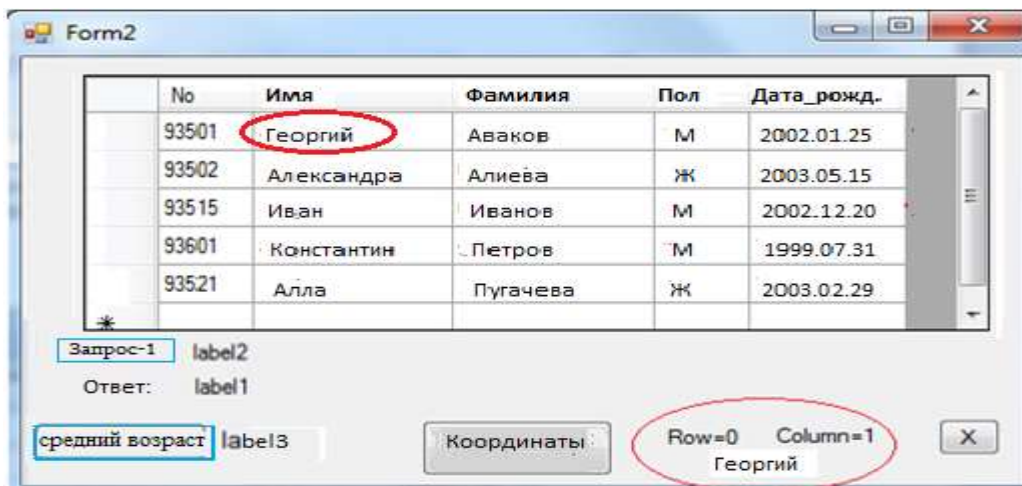


Рис.19.16-а

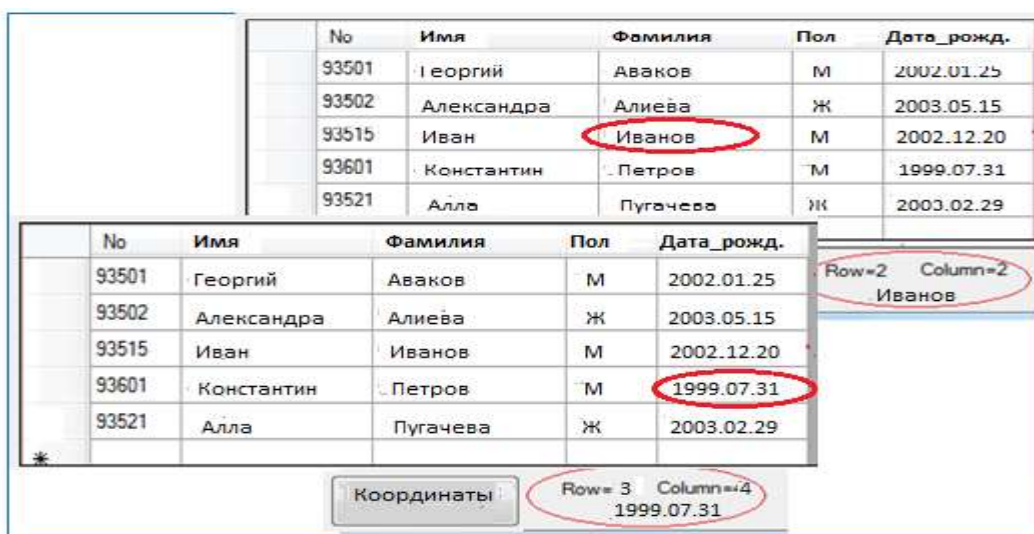


Рис.19.16-б



## Лабораторная работа N20

### 6.2. Построение C# приложения с базой данных SQL Server драйвером ADO.NET

**Цель работы:** изучение создания программных приложений совместным использованием языка визуального программирования C#, пакета баз данных Ms SQL Server и драйвера ADO.NET.

**Задача\_1:** Дана база данных Ms SQL Server (например, система экологического мониторинга Черного моря, одной из таблиц которой (Table) есть River.dbo (реки). Необходимо создать C# проект (интерфейс пользователя), который из базы выберет данные о реках в таблицу DataGridView осуществит операции Insert, Update и Delete. Должны быть использованы средства ADO.NET драйвера.

#### 6.2.1. Подготовка экспериментальной базы данных пакетом Ms SQL Server

На рис.20.1 показана исходная база данных, которая реализована Ms SQL Server пакетом. Схема соответствует иерархии таблиц базы.

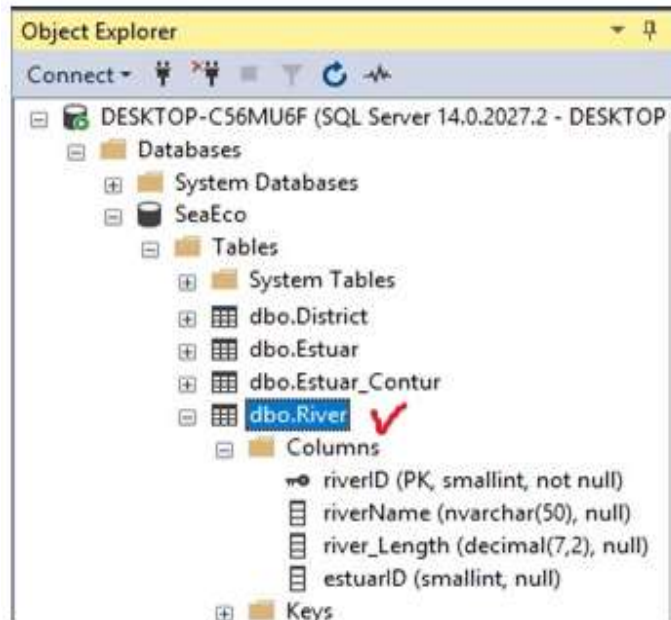


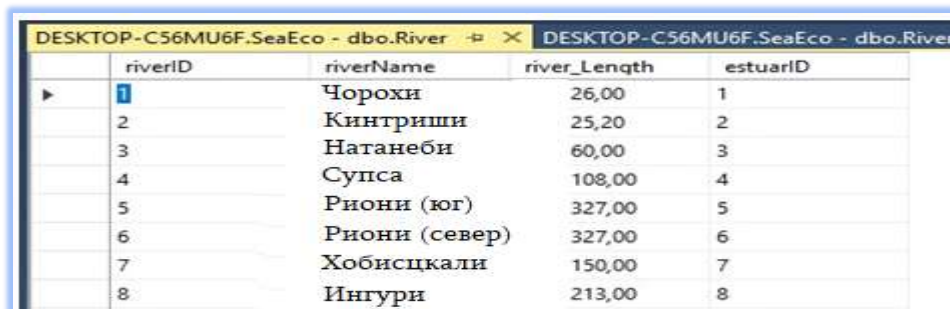
Рис.20.1. База данных Ms SQL Server-a

На рис.20.2 видна структура таблицы River с соответствующими типами данных.

Column Name	Data Type	Allow Nulls
riverID	smallint	<input type="checkbox"/>
riverName	nvarchar(50)	<input checked="" type="checkbox"/>
river_Length	decimal(7, 2)	<input checked="" type="checkbox"/>
estuarID	smallint	<input checked="" type="checkbox"/>

Рис.20.2. Структура таблицы River (Реки) в Ms SQL Server.

На рис.20.3 даны записи, ввод которых был осуществлен заранее (впрочем, для нашей программы их наличие не является обязательным. Их можно ввести интерфейсом).



riverID	riverName	river_Length	estuarID
1	Чорохи	26,00	1
2	Кинтриши	25,20	2
3	Натанеби	60,00	3
4	Супса	108,00	4
5	Риони (юг)	327,00	5
6	Риони (север)	327,00	6
7	Хобисцкали	150,00	7
8	Ингури	213,00	8

Рис.20.3. Исходная таблица River (Реки) в Ms SQL Server

### 6.2.2. Создание нового проекта на платформе .NET с использованием языка C#

На рис.20.4 показана процедура создания проекта.

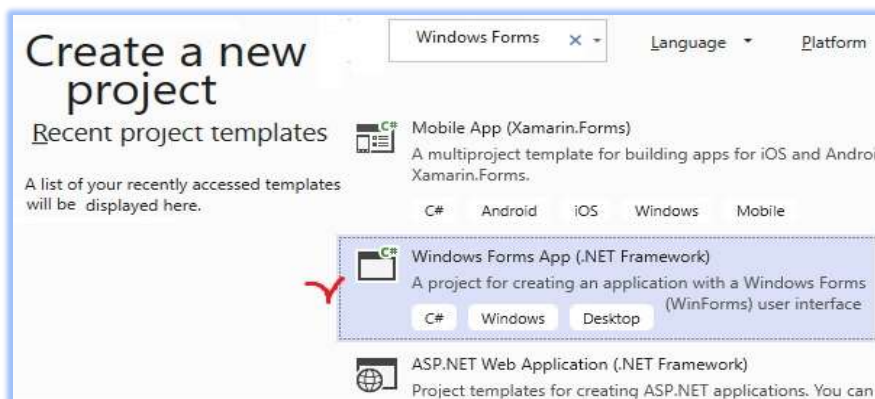


Рис.20.4-а. Создание проекта WinForm App

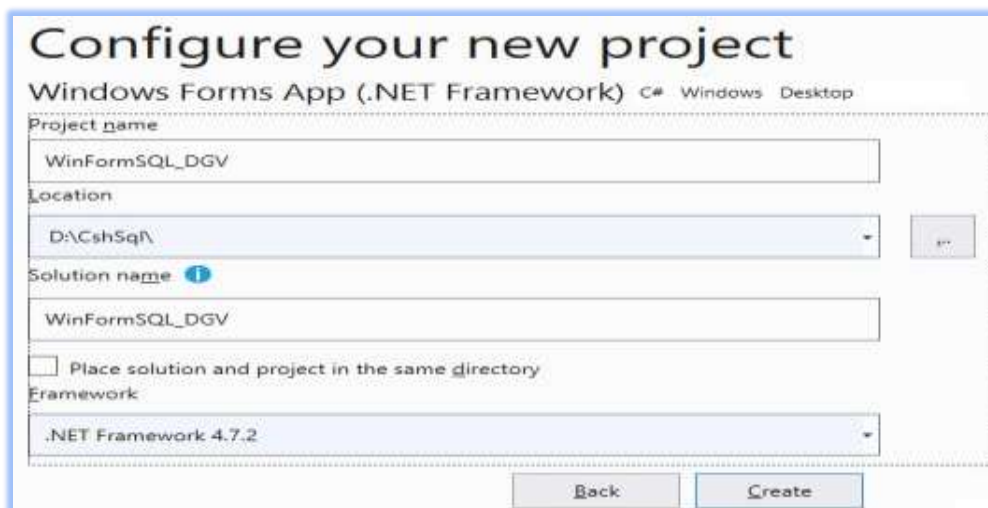


Рис.20.4-б. Размещение проекта WinFormSQL\_DGV в каталоге системы

Выбираем имя проекта (Name), место его хранения (с помощью Browse) и Solution name. Затем ОК.

### 6.2.3. Присоединение к пректу базы данных

Необходимо осуществить присоединение базы данных (Connect to Database). Это показано на рис. 20.5.

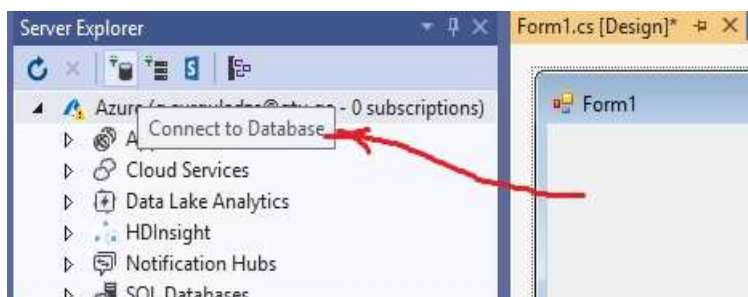


Рис.20.5. Connect to Database

Появляется новое окно (рис. 20.6), в котором надо выбрать соответствующий источник (Data Source), сервер (Server name) и базу данных (Database name).

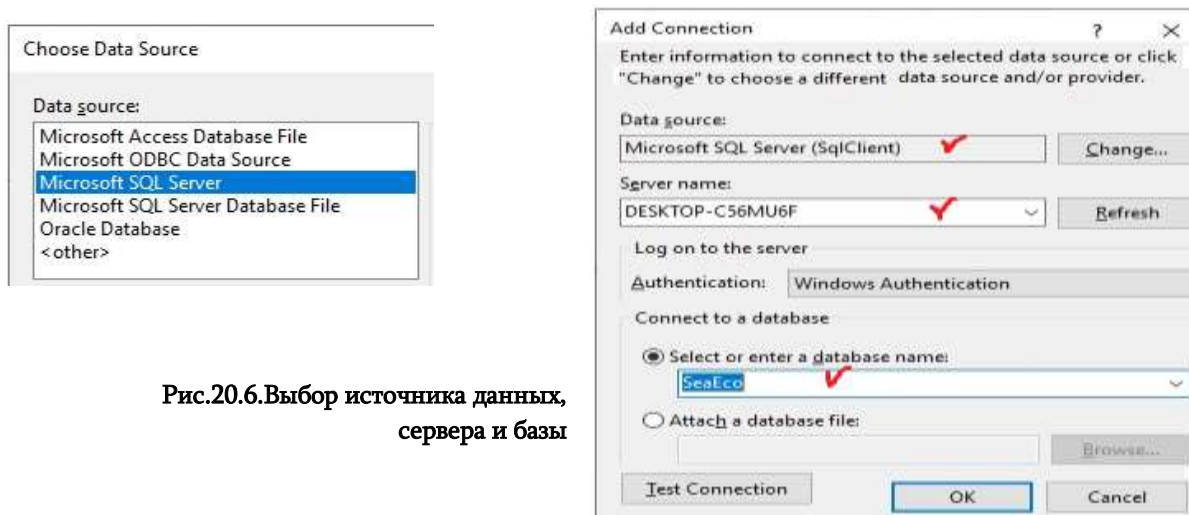


Рис.20.6.Выбор источника данных, сервера и базы

### 6.2.4. Активизация элемента DataGridView и определение параметров таблицы

Из инструментальной панели на форме перенесем элемент DataGridView и приведем в действие маленькую стрелкой верхний правый угол. Получим рис.20.7.

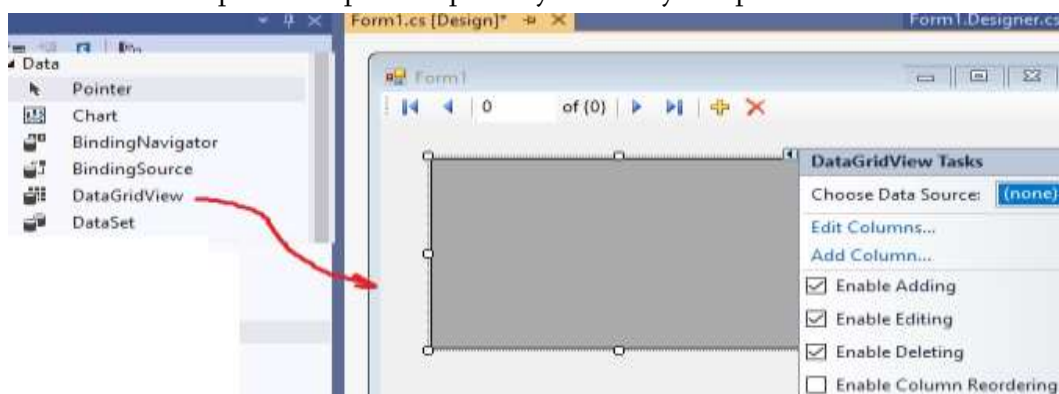


Рис.20.7. Определение параметров

На рисунке видно, что операции добавления, редактирования и стирания разрешены (чекбоксы отмечены). Выберем клавишу Choose Data Source чекбокса, получим рис.20.8.

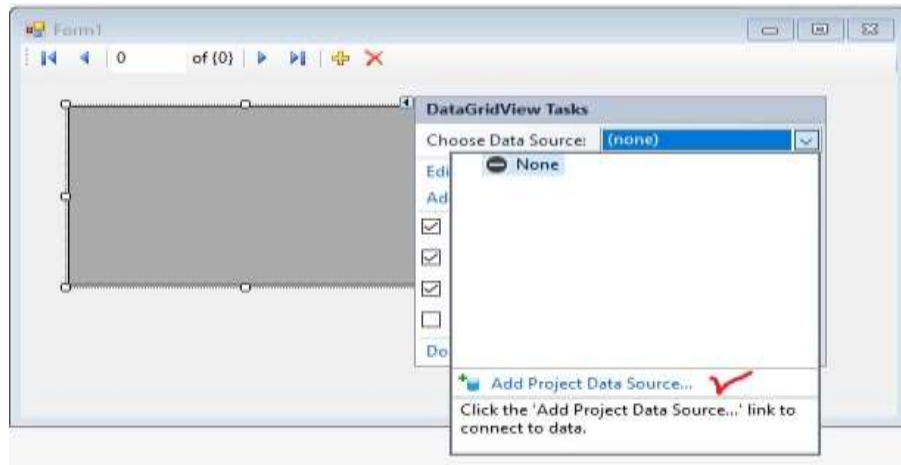


Рис.20.8. Добавление источника данных

Задействуем Add Project Data Source и перейдем на рис.20.9.

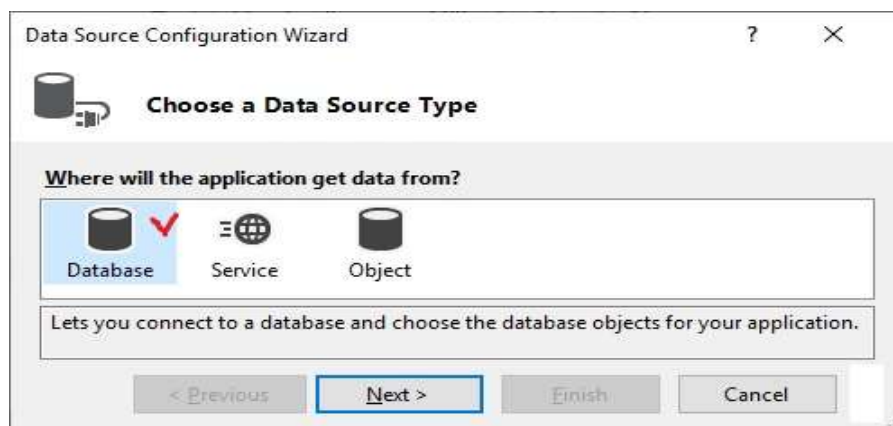


Рис.20.9. Выбор типа источника данных

Выбираем Database и Next (Рис.20.10).

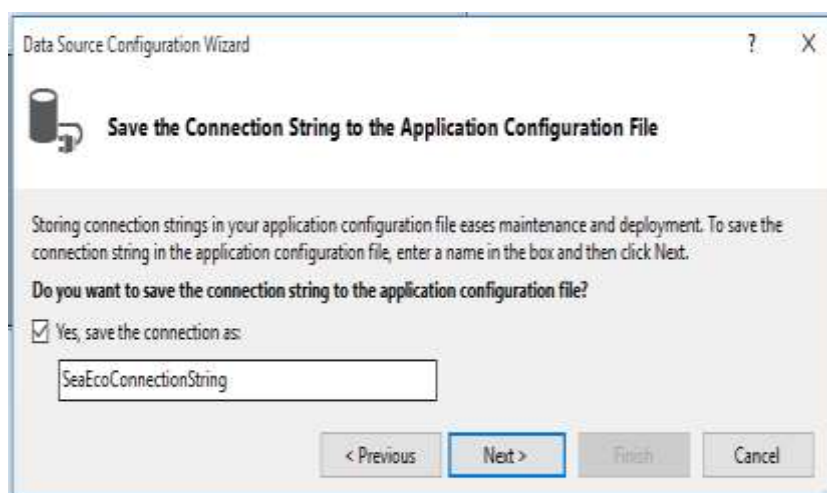


Рис.20.10. Выбор Connection (присоединения) данных

Параметр **Connection** у каждого компьютера будет свой. Его определение возможно из Server Explorer (в нашем случае параметр есть: DESKTOP – C56MU6F.SeaEco.dbo). Затем Next и переходим на рис.20.11.

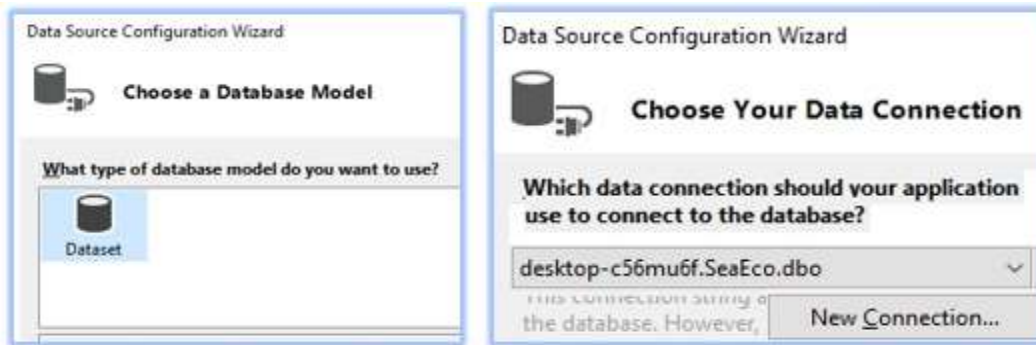


Рис.20.11. Хранение Connection String в файле конфигурации приложения

Затем вызывается окно выбора объектов базы данных (Рис.20.12).

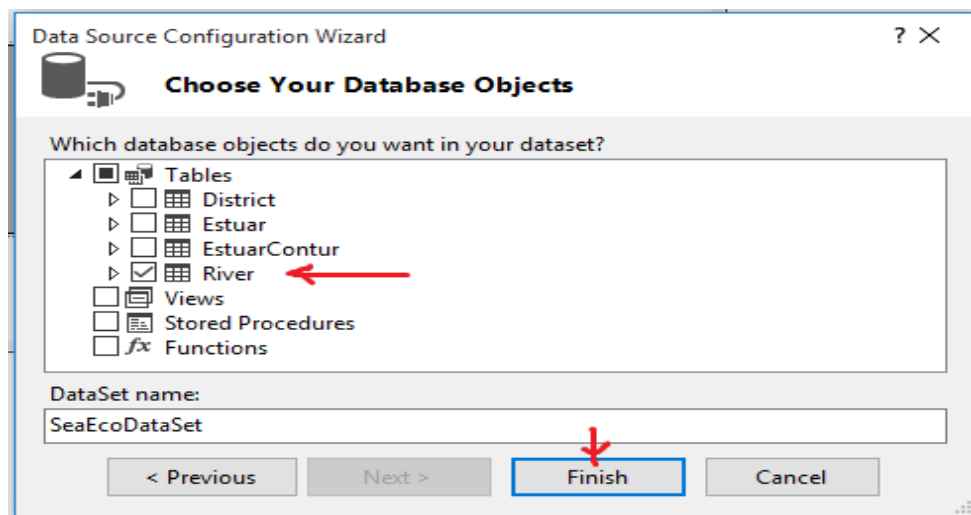


Рис.20.12. Выбор объекта (например, River).

На рис.20.13 показан результат определения значения источника данных, в нашем случае это есть: riverBindingSource.

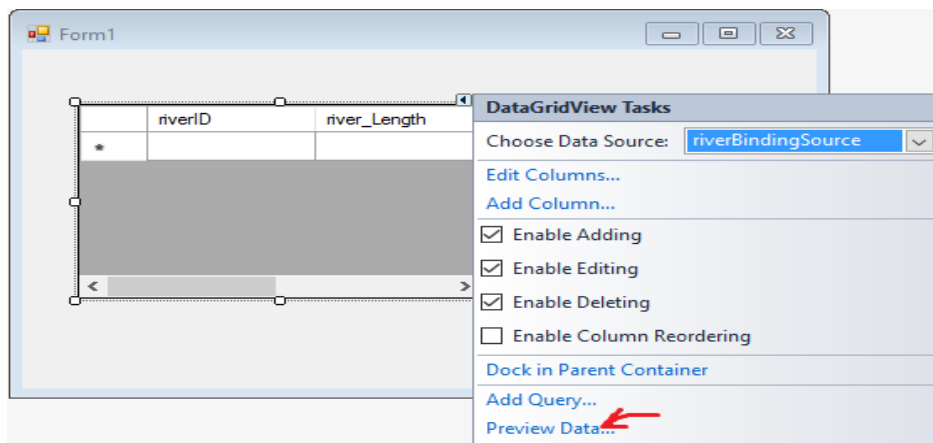


Рис.20.13. Результат Data Source:” riverBindingSource”

Здесь же можно воспользоваться линком Preview Data и посмотреть просмотреть заранее прикрепленные к базе записи таблицы Рис.20.14).

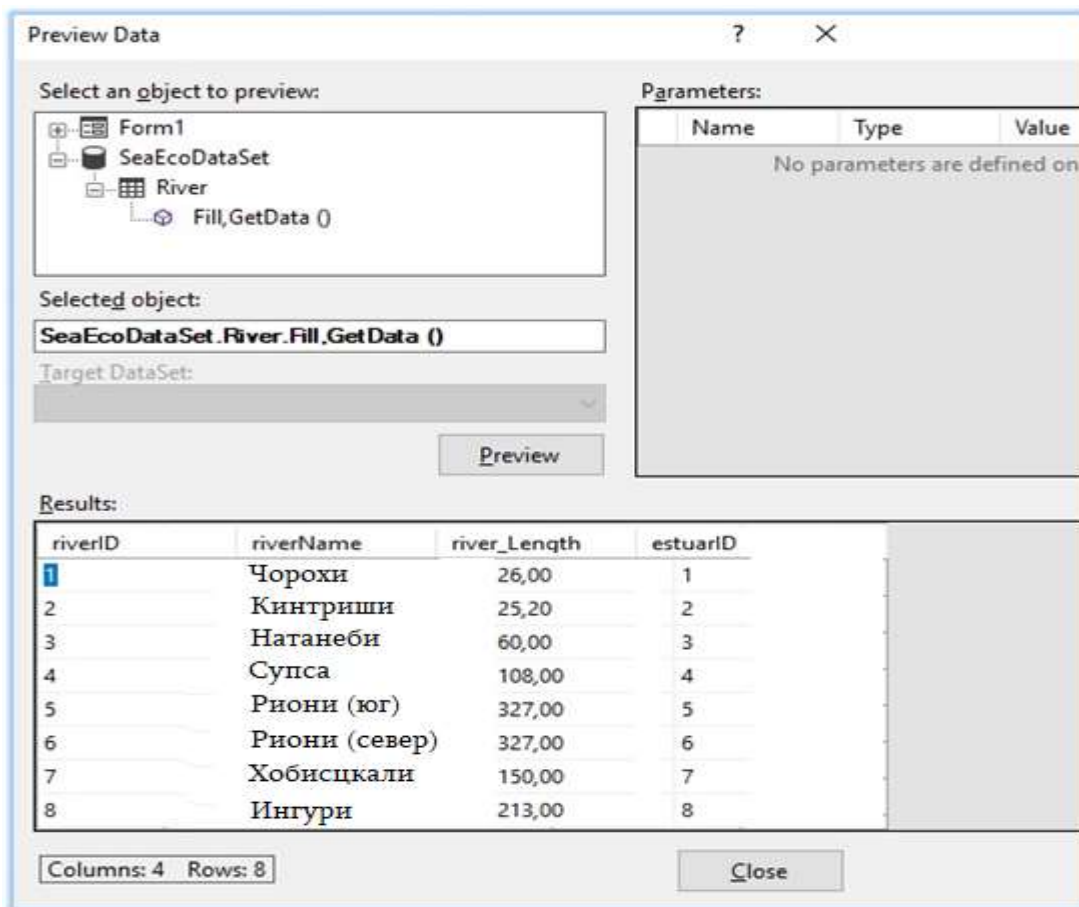


Рис.20.14. Таблица Preview Data

И наконец, в Properties-е Form1 изменим имя “Реки Черного моря (в пределах Грузии)”, из панели инструментов перенесем три клавиши (Button 1,2,3), дадим имена (Рис.20.15).

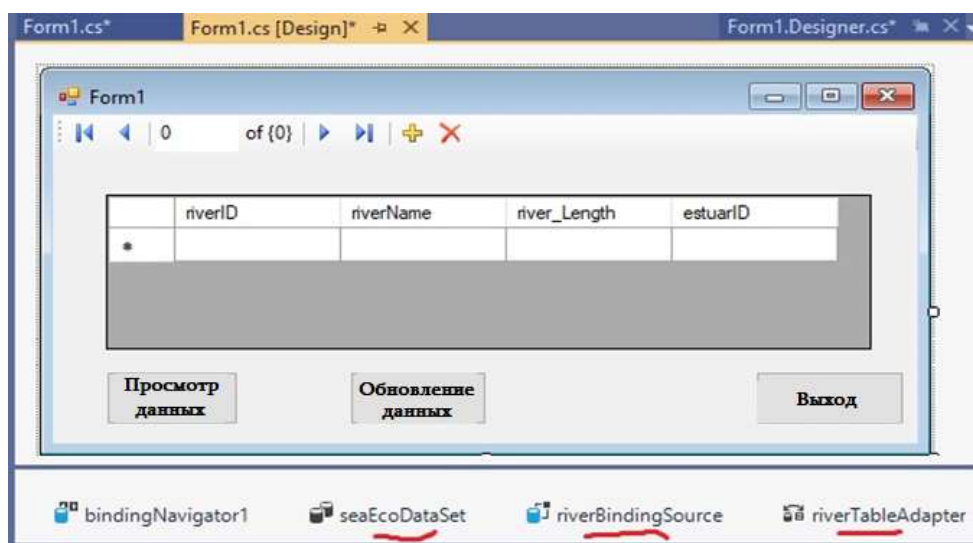


Рис.20.15. Основной интерфейс проекта

### 6.2.5. Вопросы программной реализации

Теперь перейдем на программирование клавиш интерфейса системы, т.е. должен осуществиться просмотр записей таблицы соответствующей базы SQL Server-а, *добавление, замена и удаление*.

Первоначально добавим пространство имен:

```
using System. Data.SqlClient;
```

Затем объявим глобальные переменные:

```
SqlDataAdapter sda;
```

```
SqlCommandBuilder scb;
```

```
DataTable dt;
```

Для клавиши “Просмотр данных” (button1) программный код будет:

```
private void Button1_Click(object sender, EventArgs e)
{
    SqlConnection con = new SqlConnection("Data Source=DESKTOP-C56MU6F;
        Initial Catalog = SeaEco; Integrated Security = True");
    sda = new SqlDataAdapter(@"SELECT riverID,
        river_Length, riverName,
        estuarID from River", con);

    dt = new DataTable();
    sda.Fill(dt);
    dataGridView1.DataSource = dt;}

```

После работы программы получим следующую картину (Рис.20.16).

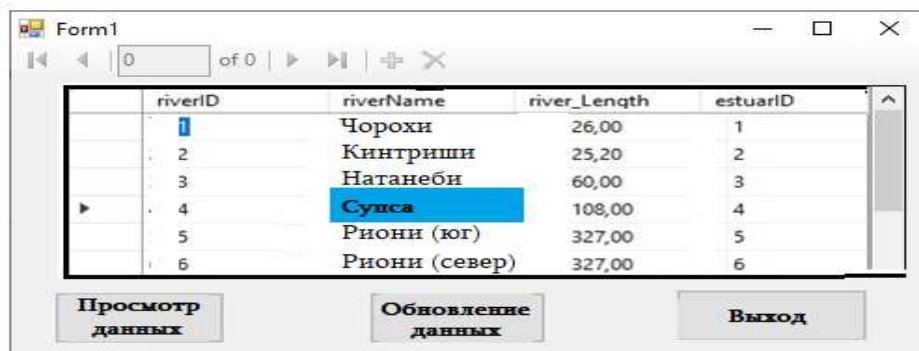


Рис.20.16. Результат, полученный после использования клавиши “просмотр данных’ (DataShow)

Код клавиши “обновление данных” (Insert, Update, Delete) показан ниже:

```
private void Button2_Click(object sender, EventArgs e)
{
    scb = new SqlCommandBuilder(sda);
    sda.Update(dt);
}

```

Полный программный код приводится в следующем листинге:

```
// --- Листинг --- Insert, Update, Delete for DataGridView_SQL----
using System;
using System.Collections.Generic;
using System.ComponentModel;
using System.Data;
using System.Drawing;
using System.Linq;
```

```

using System.Text;
using System.Threading.Tasks;
using System.Windows.Forms;
using System.Data.SqlClient; // !!!
namespace WinFormSQL_DGV
{
publicpartialclassForm1 : Form
    {
        SqlDataAdapter sda;
        SqlCommandBuilder scb;
        DataTable dt;
publicForm1()
    {
        InitializeComponent();
    }
privatevoid Form1_Load(object sender, EventArgs e)
    {
        // TODO: This line of code loads data into the 'seaEcoDataSet.River' table.
        // You can move, or remove it, as needed.
        this.riverTableAdapter.Fill(this.seaEcoDataSet.River);
    }
// ---- Просмотр данных ----
privatevoid Button1_Click(object sender, EventArgs e)
    {
        SqlConnection con = new SqlConnection("Data Source=DESKTOP-C56MU6F; Initial
            Catalog = SeaEco; Integrated Security = True");
        sda = new SqlDataAdapter(@"SELECT riverID,
            river_Length, riverName,
            estuarID from River", con);
        dt = new DataTable();
        sda.Fill(dt);
        dataGridView1.DataSource = dt;
    }
// ---- обновление данных ----
privatevoid Button2_Click(object sender, EventArgs e)
    {
        scb = new SqlCommandBuilder(sda);
        sda.Update(dt);
    }

privatevoid Button3_Click(object sender, EventArgs e)
    {
        Close();
    }
}
}

```

На рис.20.17 приводится изменение ив таблице двух значений (новые числа показаны в кружочках, затем с помощью клавиш 1 и 2 в базу запишутся эти измененные значения (можно закрыть программу и заново ее активизировать).

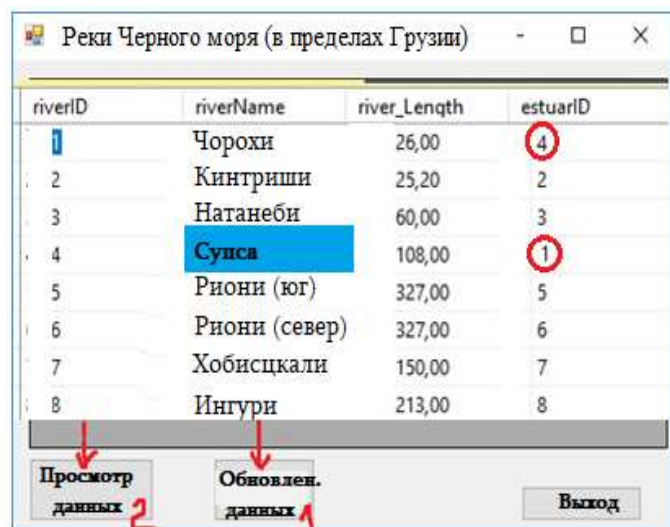


Рис.20.17. Осуществление изменений Update



На рис.20.18 показано добавление (Insert) новой строки.

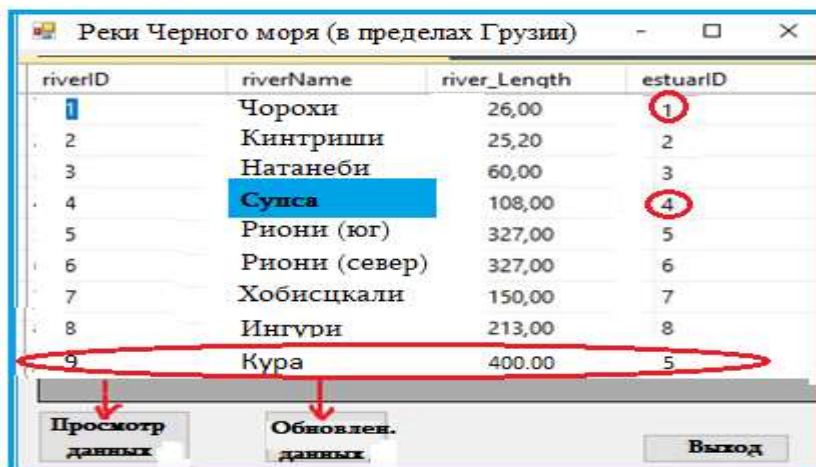


Рис.20.18. Осуществление операции Insert

В конце показано удаление (Delete) строки. Для этого пометим соответствующую строку, например, “ID – реки” и затем клавишей компьютера Delete удалим (рис.20.19).

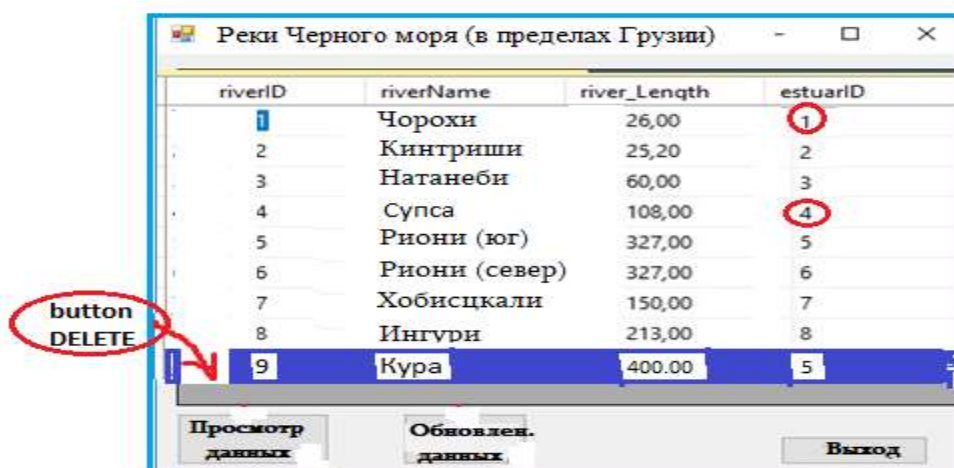


Рис.20.19. Осуществление операции Delete

На этом закончим рассмотрение задач, связанных с осуществлением связи с базой данных с помощью элементов языка C#. С их помощью был построен фрагмент информационной системы экономониторинга Черного моря.

➤ **Самостоятельная работа:**

Построить для обработки записей базы данных SQL Server используйте инструмент BindingNavigator, исследуйте его функции. Постройте интерфейс какой-либо проблемной области (Университет, библиотека, аптека, гостиница и т.д.).

## Глава 7

### Тестирование и рефакторинг программной апликации

#### Лабораторная работа N21

##### 7.1. Модульное тестирование программ

**Цель:** изучение тестирования проагммных кодов интегрированной области Visual Studio .NET Framework.

Unit testing и Coded UI являются инструментами тестирования Microsoft-а, которые выполняются в области Studio.NET.

Unit testing – модульное тестирование процесса программирования, с помощью которого проверяется корректность отдельных модулей исходного кода. Идея тестирования состоит в том, что для каждой нетривиальной функции или метода надо написать тест. Это обеспечивает быструю проверку теста, не привели ли последние изменения кода к регрессии программы, т.е. к остановке программы в уже протестированных частях.

Тест Coded UI автоматически пишет, пускает на выполнение проверяет тест-кейсы. Писать такие тесты на C# или Visual Basic-е в области Visual Studio. Технологию тестирования Unit testing рассмотрим на примере виртуального объекта, например банка.

- Создание проекта программы тестирования: в Visual Studio необходимо выбрать **File -> New -> Project**. В результате появится диалоговое окно **Visual C# => ClassLibrary** в котором выбираем проект с именем **Bank**. Получим окно **Solution Explorer**, показанное на рис.21.1. Здесь имя **Class1.cs** заменяем на **BankAccount.cs**.

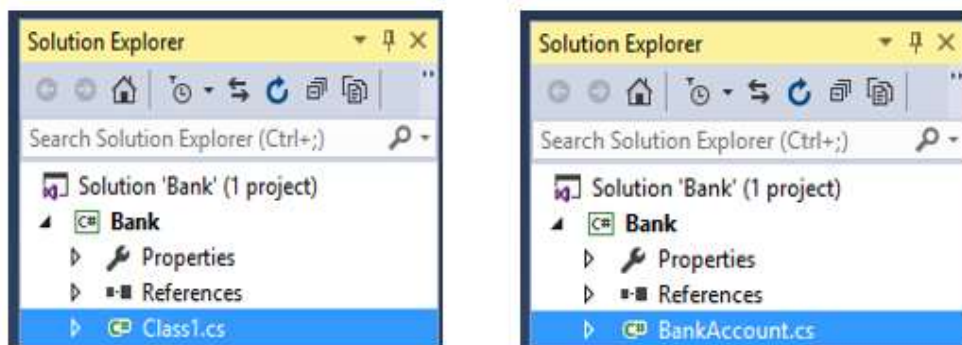


Рис.21.1. Class1 - > BankAccount.

Затем текст **BankAccount.cs** в области редактора заменим кодом нашей программы тестирования. Этот исходный текст приведен в листинге

```
//-- Листинг --- BankAccount.cs -----
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using System.Threading.Tasks;

namespace BankAccountNS
{
    public class BankAccount
    {
```

```
private string m_customerName;
private double m_balance;
private bool m_frozen = false;
private BankAccount()
{
}
public BankAccount(string customerName, double balance)
{
    m_customerName = customerName;
    m_balance = balance;
}
public string CustomerName
{
    get { return m_customerName; }
}
public double Balance
{
    get { return m_balance; }
}
public void Debit(double amount)
{
    if (m_frozen)
    {
        throw new Exception("Account frozen");
    }
    if (amount > m_balance)
    {
        throw new ArgumentOutOfRangeException("amount");
    }
    if (amount < 0)
    {
        throw new ArgumentOutOfRangeException("amount");
    }
    m_balance += amount; // განზრახვარასწორიკოდი
    // m_balance -= amount; // გასწორებული
}
public void Credit(double amount)
{
    if (m_frozen)
    {
        throw new Exception("Account frozen");
    }
    if (amount < 0)
    {
        throw new ArgumentOutOfRangeException("amount");
    }
    m_balance += amount;
}
private void FreezeAccount()
{
    m_frozen = true;
}
private void UnfreezeAccount()
{
    m_frozen = false;
}
public static void Main()
{
}
```

```

BankAccount ba = new BankAccount("Mr.Bryan Walton",11.99);
ba.Credit(5.77); ba.Debit(11.22);
Console.WriteLine("Current balance is ${0}", ba.Balance);
    }
}
}

```

- Построение тест-файла проекта (Unit Test Project):

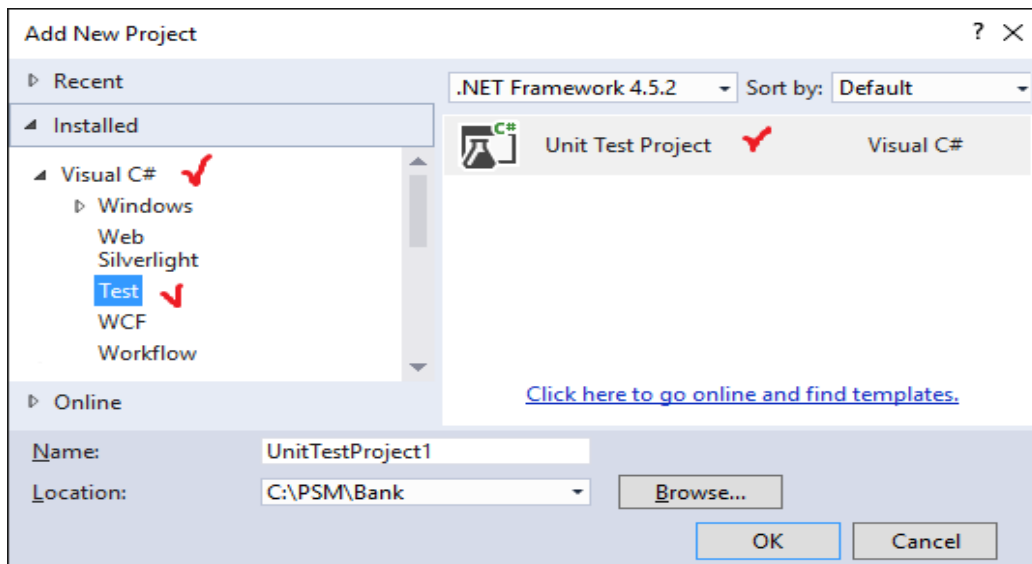


Рис.21.2. Создание проекта Unit Test

Получим картину, показанную на рис 21.3 с проектом Bank Tests. На правом рисунке имя класса UnitTest заменено на BankAccountTests.

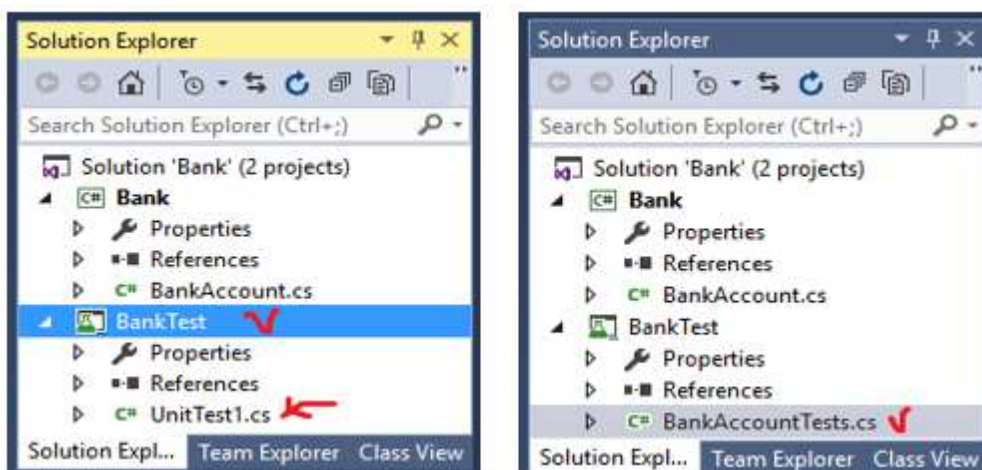


Рис.21.3

В проект **BankTest** добавим reference из **Bank** solution. Для этого из **BankTest** правой клавишей мыши выбираем **Add Reference** и получим окно, показанное на рис.21.4. Здесь в строке Solution выбираем Projects и пометим чекбокс Bank-а.

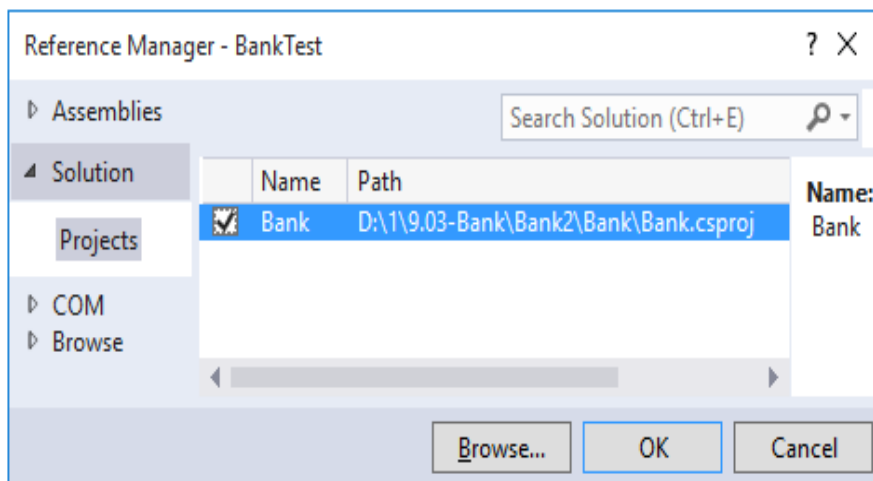


Рис.21.4

Получим результат, показанный на рис.21.5.

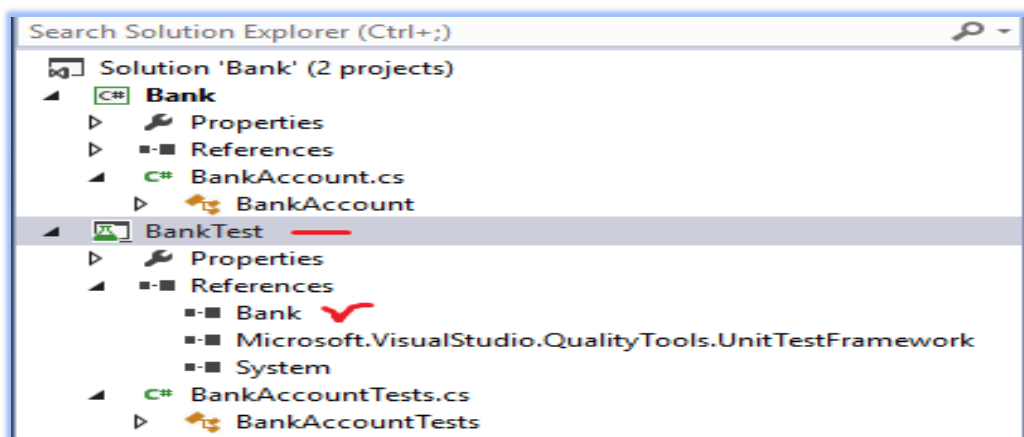


Рис.21.5

Добавим в программу BankAccountTest из проекта Bank область имен: using BankAccountNS.

Таким образом, файл BankAccountTest.cs будет иметь вид, показанный в листинге:

```
//-- Листинг----- BankAccountTests.cs -----
using System;
using Microsoft.VisualStudio.TestTools.UnitTesting;
using BankAccountNS;

namespace UnitTestProject1
{
    [TestClass]
    public class BankAccountTests
    {
        [TestMethod]
        public void TestMethod1()
        {
        }
    }
}
```

```
}  
}
```

Теперь создадим первый тест-метод. В этой процедуре написаны методы унит-теста `BankAccount class`-а для метода `Debit` верификации поведения. Эти методы показаны выше.

Путем анализа методов тестирования выяснилось, что проверить минимум три метода поведения:

1. Метод создает исключение `ArgumentOutOfRangeException`, если сумма кредита превысит баланс;
2. Он создает исключение `ArgumentOutOfRangeException` даже тогда, когда величина кредита отрицательная;
3. Если пункты 1 и 2 успешно завершились, тогда метод подсчитывает сумму из баланса.

В первом тесте проверим, что при допустимой величины кредита со счета будет списана необходимая сумма (когда она имеет положительное значение и меньше на балансовом счете).

Добавим в класс `BankAccountTests` следующий метод:

```
// ---- unit test code ----  
[TestMethod]  
public void Debit_WithValidAmount_UpdatesBalance()  
{  
    // arrange  
    double beginningBalance = 11.99;  
    double debitAmount = 4.55;  
    double expected = 7.44;  
    BankAccount account = new BankAccount("Mr. Dito", beginningBalance);  
    // act  
    account.Debit(debitAmount);  
    // assert  
    double actual = account.Balance;  
    Assert.AreEqual(expected, actual, 0.001, "Account not debited correctly");  
}
```

Метод довольно прост. Мы создаем новый объект `BankAccount` с исходным балансом и затем вычитываем правильную величину. Мы используем фреймворк унит-теста `Microsoft`-а для управляемого кода метода `AreEqual`, чтобы завершилась верификация конечного баланса.

У тест-метода следующие требования:

1. Метод должен быть отмечен атрибутом `[TestMethod]`;
2. Метод должен вернуть `void`;
3. Метод не должен иметь параметры.

Код теста приведен в листинге:

```
//-- Листинг----- BankAccountTests.cs ----  
using System;  
using Microsoft.VisualStudio.TestTools.UnitTesting;  
using BankAccountNS;  
  
namespace UnitTestProject1  
{
```

```

[TestClass]
public class BankAccountTests
{
    [TestMethod]
    public void Debit_WithValidAmount_UpdatesBalance()
    {
        // arrange
        double beginningBalance = 11.99;
        double debitAmount = 4.55;
        double expected = 7.44;
        BankAccount account = new BankAccount("Mr. Dito",
beginningBalance);
        // act
        account.Debit(debitAmount);

        // assert
        double actual = account.Balance;
        Assert.AreEqual(expected, actual, 0.001, "Account
not debited correctly");
    }
}

```

- из меню BUILD выбираем Build Solution;
- из меню TEST выбираем пункты Windows и Test Explorer. Открывается окно Test Explorer (Рис.21.6).

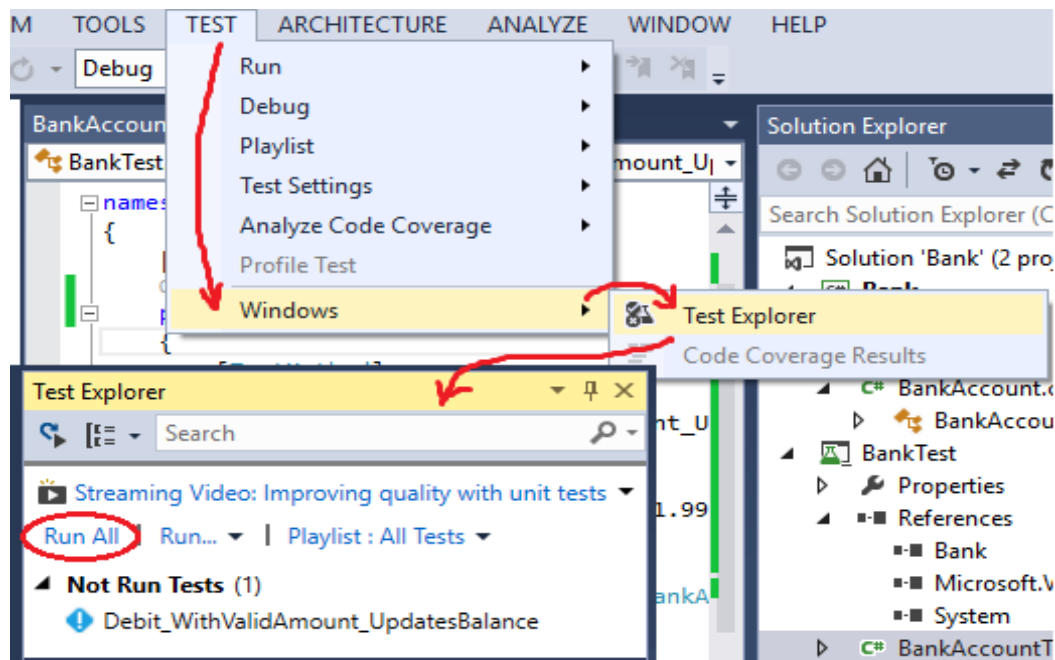


Рис.21.6

Здесь выбираем Run All и получаем результат (Рис.21.7).

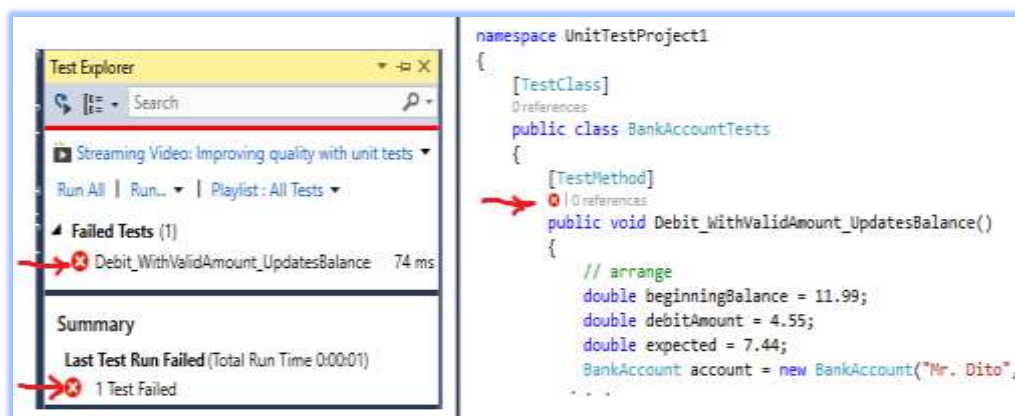


Рис.21.7

На рисунке символы “x” помещены в красные кружочки, т.е. тестирование выявило ошибки и код результативно не завершился.

Если бы метод завершился успешно, тогда получим зеленные x-символы.

Следующим этапом является исправление кода и повторное тестирование. В тестируемой программе заменим в строке символ “+” на “-”.

```
// m_balance += amount; // intentionally incorrect code
m_balance -= amount; // intentionally correct code
```

После повторной работы теста получаем успешный результат, т.е. получаем символы зеленого цвета (Рис.21.8).

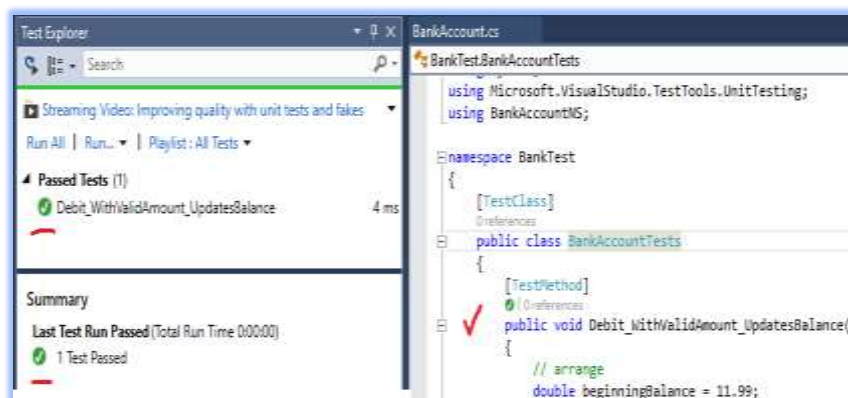


Рис.21.8

### ➤ Самостоятельная работа:

- Создайте новый проект WinFormsApp\_Test, в котором будет тестируемая программа;
- Написать тестируемый код в программной части проекта (согласно заданию данной лаб.) - BankAccount.cs;
- Построить проект программы тестового файла (Unit Test Project);
- Скорректируйте код BankAccountTests, добавив соответствующее пространство имен и ссылку;
- Создайте 3 тестовых метода (исключения) для проверки поведения;
- Отладить и запустить тест;
- Проанализировать полученные результаты, выявить ошибки тестирования (красные отметки);
- Исправьте код и снова запустите его для тестирования;
- 9. Правильные результаты отмечены зелеными метками.



## Лабораторная работа №22

### Рефакторинг: обработка и реорганизация кода

**Цель работы:** изучение процедур разработки и модификации программного кода на C#, ознакомление со смыслом “рефакторинга” и его применение с целью обновления существующей версии кода.

Обработка программного кода осуществляется в соответствии с выбранным типом проекта и шаблона кода, которую позволяет осуществить рабочая область Visual Studio. NET. В тоже время осуществляется проверка корректности синтаксиса исходного, автоматическое завершение генерирования кода, применение в коде процесса навигации (например, из контекстного меню с помощью Go to definition).

Все это значительно повышает производительность работы программиста – деvelopepa !

“Рефакторинг” есть систематическая модификация и усовершенствование существующего программного кода, без коренного изменения семантики функционирования. Внесение изменений в код осуществляется автоматизированным обновлением, которое позволяет осуществить рабочая область .NET.

Типичный пример этого – **изменение имени метода**. Задача заключается в осуществлении изменения имени и его определений какого-либо конкретного метода, при этом во всех местах его применения.

Если проект довольно большой, то осуществление таких изменений **вручную** задача трудоемкая и неудобная. Не исключена ситуация, что в каком-либо модуле забудем осуществить изменение, что в дальнейшем, во время работы программы проявится в виде ошибки и придется заново корректировать проект.

Рассмотрим задачу рефакторинга для прстейшей консольной аппликации (рис.22.1).

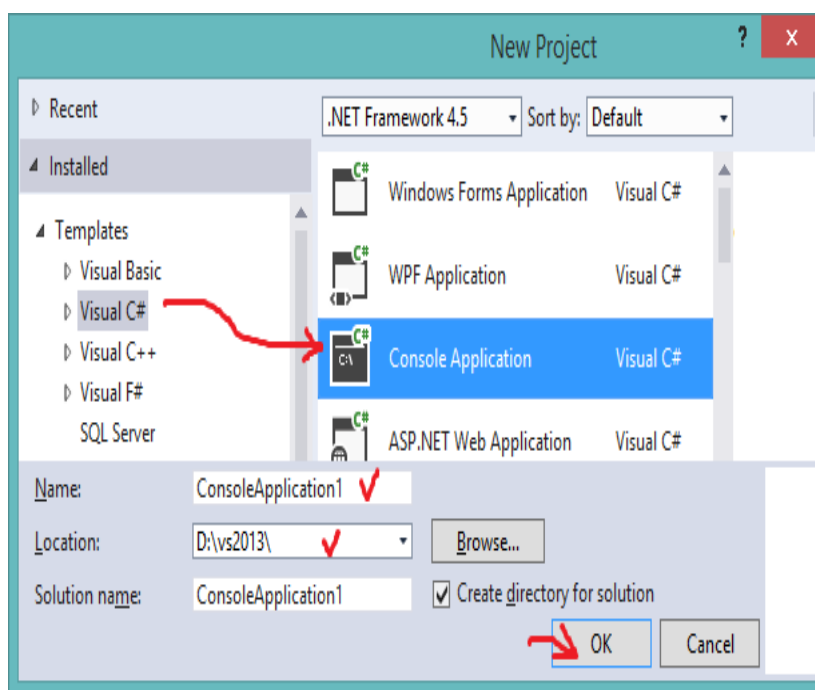


Рис.22.1.Создание консольной аппликации

В исходный текст программы добавим несколько строк, что показано на рис 22.2.

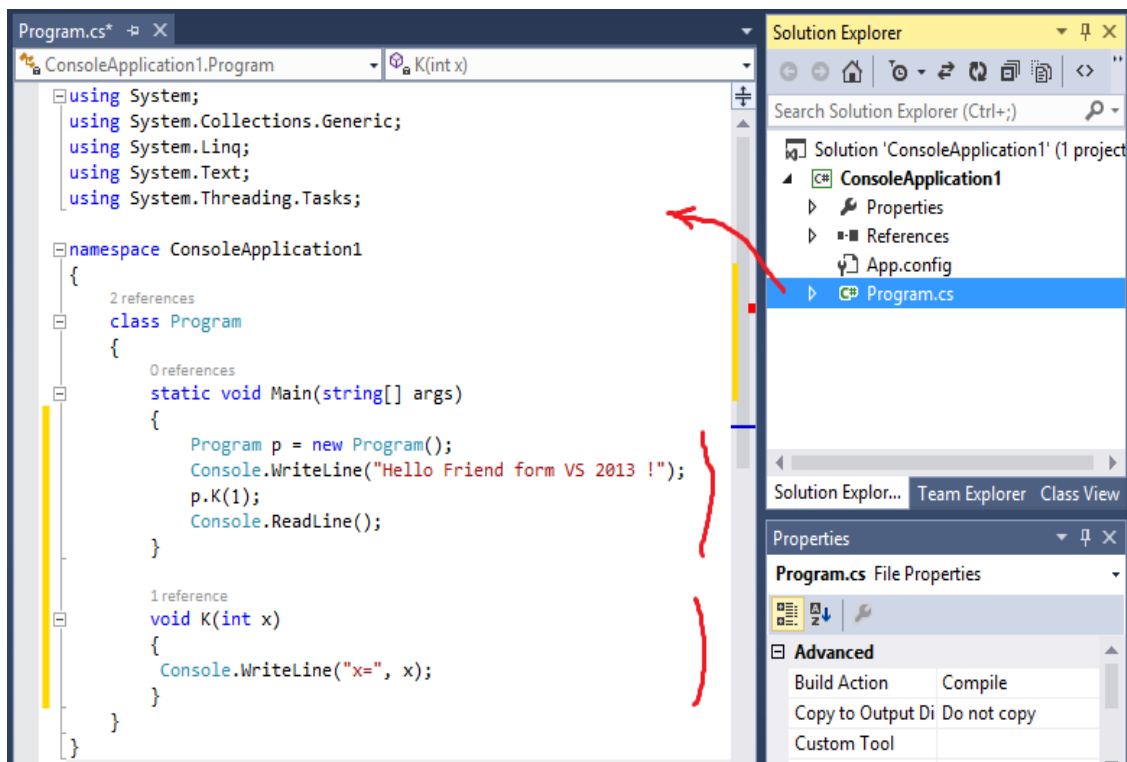


Рис. 22.2

В программе определен класс Program, статический метод Main и метод *K* - экземпляра с аргументом x. Предположим, необходимо изменить название метода *K* на *M*. Изменение должно осуществиться не только в определении метода, Но и во всех местах его использования !

В области корректировки кода курсор мыши применяем в первой строке *K* метода, пометим слово, которое надо изменить и из контекстного меню выбираем пункт Refactor / Rename (рис.22.3).

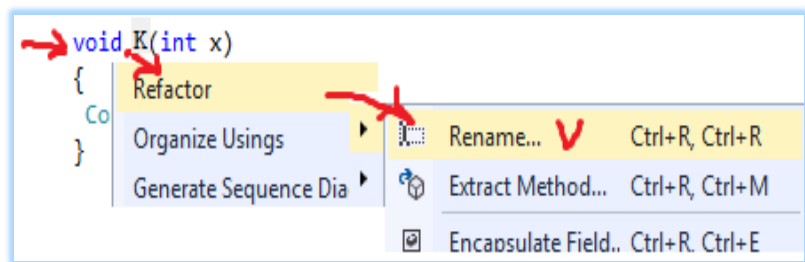


Рис.22.3. Выбор действия для K метода рефакторинга

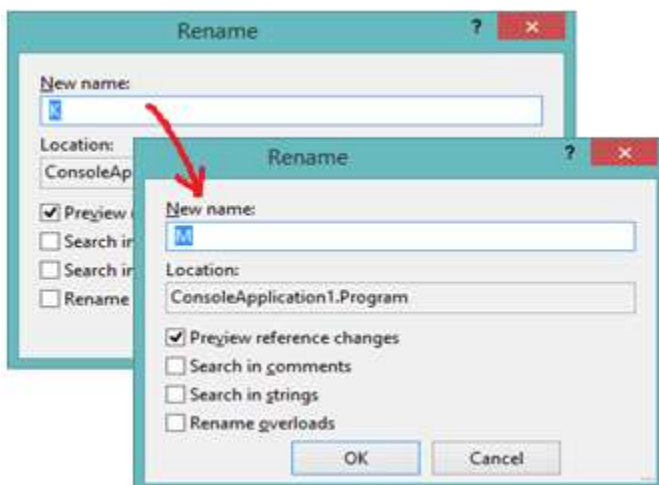


Рис.22.4. K заменяется M -ом.

Получаем рис.22.5.

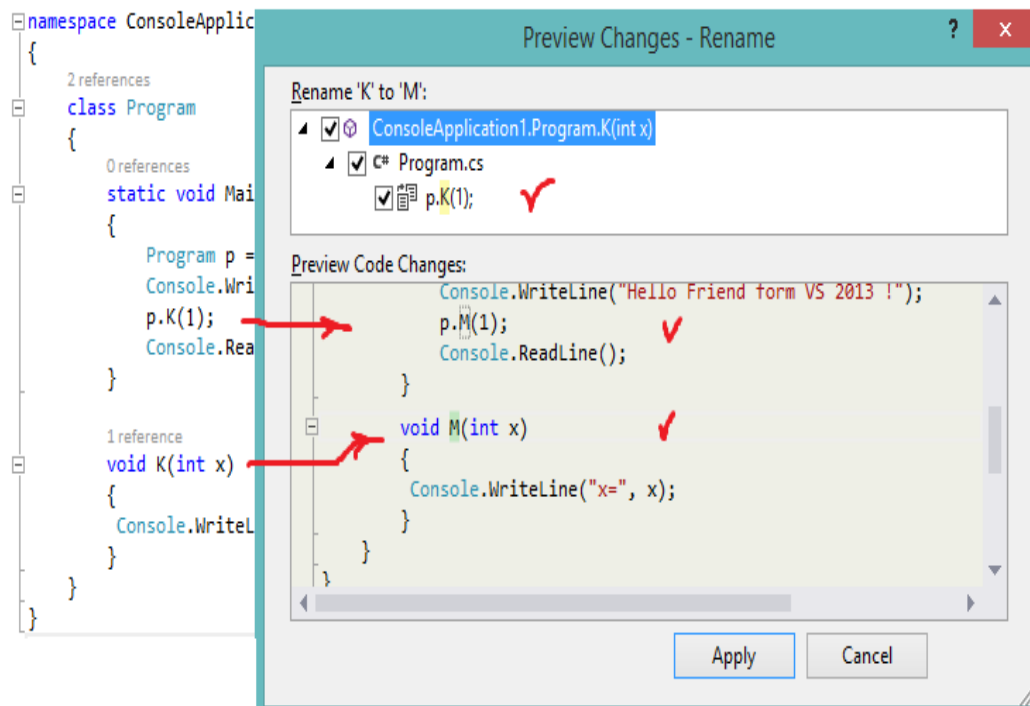


Рис.22.5. Предварительный просмотр запланированных изменений (Preview Changes)

Если все в порядке, тогда выбираем Apply и получаем итог рефакторинга (рис.22.6).

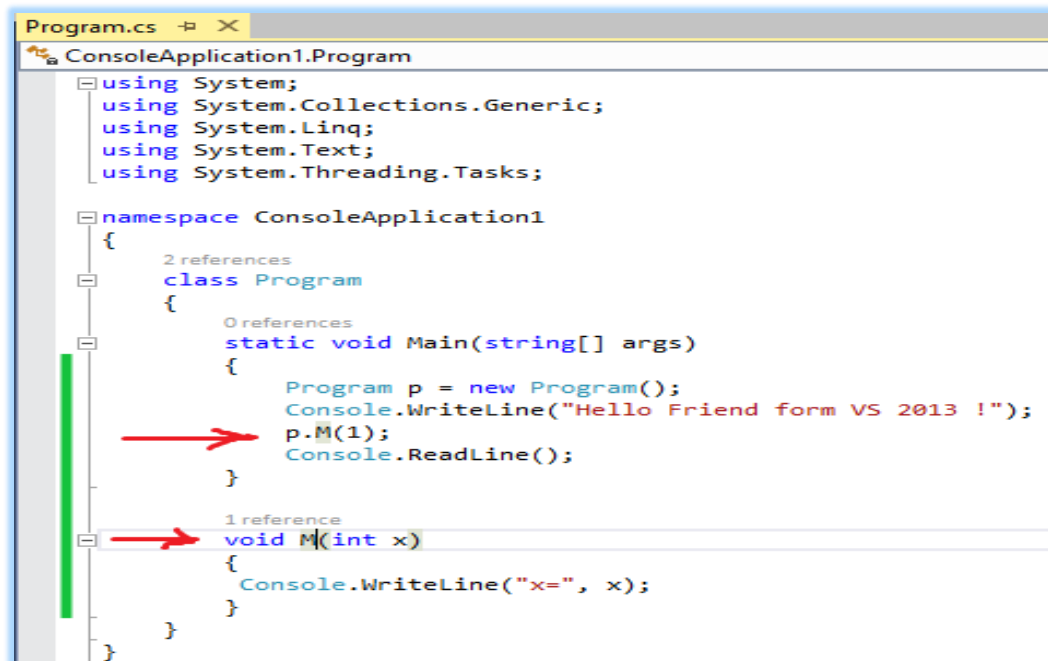


Рис.22.6. Итог рефакторинга

Рабочая область VS/.NET имеет следующие возможности рефакторинга:

- Rename - изменение имени;
- Extract Method – Экстракт (извлечение) метода: модификация отмеченного фрагмента кода по имени, указанному в методе;
- Encapsulate Field - Инкапсулирование поля, преобразованием его в приват, но добавлением свойства с целью его доступности;
- Extract Interface - Экстракт (извлечение) интерфейса: пометка текста класса для автоматического формирования соответствующего ему интерфейса (если это возможно);
- Remove parameters - стирание части параметров метода;
- Reorder parameters - изменени порядка параметров метода.

### ➤ Самостоятельная работа:

Постройте аппликацию простейшей консоли в рабочей области Visual Studio. NET (например: университет, супермаркет, аптека, производство\_продукции\_поставка или др.). Проведите процедуры рефакторинга и исследуйте возможности вышеперечисленных его разновидностей.

БИБЛИОГРАФИЧЕСКИЙ СПИСОК

1. Долженко А.И., Глушенко С.А.. Программная инженерия. Учебное пособие. Ростов-на-Дону. 2017. -128 с.
2. Sommerville I. Software engineering, 11th ed., ISBN 978-0137035151. Addison-Wesley. 2016
3. Сургуладзе Г., Туркия Э. Основы управления программными системами. ГТУ, "Технический ун-т". Тб., 2016. -350 с.
4. Booch G., Jacobson I., rambaugh J. (1996). Unified Modeling Language for Object-Oriented Development. Rational Software Corporation, Santa Clara,
5. Бек К. Шаблоны реализации корпоративных приложений. Экстремальное программирование: Пер. с англ. М.: Вильямс. 2008
6. Кознов Д.В. Введение в программную инженерию. 2009. <http://www.intuit.ru/departement/se/inprogeng>.
7. Scrum.org. <https://www.scrum.org/Resources>
8. Петкович Д. Microsoft SQL Server 2012. Руководство для начинающих: Пер. с англ. - СПб.: БХВ-Петербург. 2013
9. Surguladze G., Gavardashvili A., Topuria N. (2016). Determination of the Ecological Parameters of the Black Sea and Designing its Multimedia Base based on the Object-Role Modeling. XXVII Intern.Scientific Conf., `Problems of Decision Making under Uncertaintie`. Kiev, pp.65-68.
10. Wegner P. Concepts and paradigms of object-oriented programming. {OOPS messenger}, 1(1): 7-87, August '90.1990
11. Perkins B., Hammer J.V., Reid J.D. (2016). Beginning Visual C# 2015 Programming. Published by John Wiley & Sons, Inc. Indianapolis, IN 46256 . Copyright © 2016.
12. Скопин И. (2004). Основы менеджмента программных проектов. Новоси-бирский Гос.Унив., Россия '04. INTUIT. <http://www.intuit.ru/~studies~/courses/38/38/info>
13. Сургуладзе Г., Туркия Э. Основы программной инженерии. ISBN 978-9941-8-3808-8 Справочное руководство. «Научный центр IT-консалтинг», Том, 2022, - 250 с.
14. Сургуладзе Г. Методы и методологии компьютерного программирования (SP, OOP, VP, Agile, UML). ISBN 978-9941-1900-1. ГТУ, "ИТ-консалтинговый центр". Тб., 2019. -200 с.
15. Сургуладзе Г., Петриашвили Л. Визуальное программирование на языке C# для информационных систем (платформа Visual StudioNET 2019). ISBN 978-9941-8-1708-3. ГТУ, "ИТ-консалтинговый центр". Тб., 2019. М5, -200 с.

Передан в производство 15.03 2023 г. Размер офсетной бумаги 60X84 1/16.  
Условная печатная форма 6.5. Тираж 50 экз.



«Научный центр ИТ-консалтинга» ГТУ,  
Тбилиси, М. Костава 77

ISBN 978-9941-8-5110-0

