МИНИСТЕРСТВО НАУКИ И ВЫСШЕГО ОБРАЗОВАНИЯ РОССИЙСКОЙ ФЕДЕРАЦИИ

федеральное государственное бюджетное образовательное учреждение
высшего образования

«УЛЬЯНОВСКИЙ ГОСУДАРСТВЕННЫЙ ТЕХНИЧЕСКИЙ УНИВЕРСИТЕТ»

Кафедра «Измерительно-вычислительные комплексы»

В.В. Родионов

ТЕХНОЛОГИИ ПРОГРАММИРОВАНИЯ

Часть 1

лабораторный практикум для студентов направления
09.03.02 «Информационные системы и технологии»

Ульяновск, 2024

Лабораторная работа № 1. *Интегрированная среда разработки Visual Studio. Типы данных языка C#. Формы и элементы управления Windows Forms*

1. Интегрированная среда разработки Visual Studio

Материал рассматривается на примере **Visual Studio Express 2012** и платформы **.NET Framework**, но применим и к более новым версиям **Visual Studio**.

1.1 Навигация в интегрированной среде разработки

Разработчик может перемешаться между открытыми окнами интегрированном среды разработки без помощи мыши. **Visual Studio Express 2012** предоставляет здесь несколько вариантов. Для перемещения по открытым окнам вперед (слева направо) можно использовать комбинацию клавиш **Ctrl** + **-** (знак «минус»). Для перехода в обратном направлении (справа налево) можно использовать комбинацию клавиш **Ctrl** + **Shift** + **-**.

Аналогичных результатов можно добиться путём использования визуального средства, которое называется *навигатором* (**Navigator**). Для доступа к нему нужно использовать комбинацию клавиш **Ctrl + Tab** (или **Ctrl + Shift + Tab**). При этом появится окно с списками активных инструментов и файлов (рис. 1). Перемещаться между списками можно при помощи клавиш → и ←.

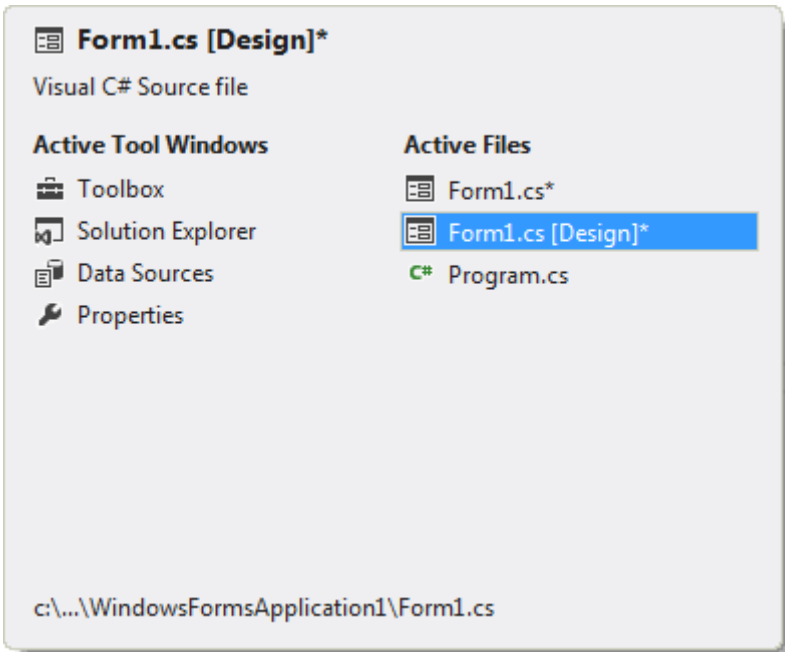

Рис. 1. Окно навигатора

1.2 Конфигурирование среды разработки

После установки **Visual Studio Express 2012** настройки среды устанавливаются в значения по умолчанию и при работе могут изменяться по усмотрению разработчика. Для управления этими настройками предназначен инструмент **Import and Export Settings**, вызываемый из

меню **TOOLS**. Мастер позволяет импортировать настройки, экспортировать их в файл или восстанавливать настройки в один из вариантов, предлагаемых интегрированной средой.

При экспорте или импорте настроек можно указать, какие из них следует применить. На рис. 2 изображён третий шаг мастера импорта настроек.

Также возможна работа с собственными коллекциями настроек, которые сохраняются в файле с расширением **vssettings**. Его можно использовать для миграции настроек с одного компьютера на другой и с одной версии интегрированной среды на другую.

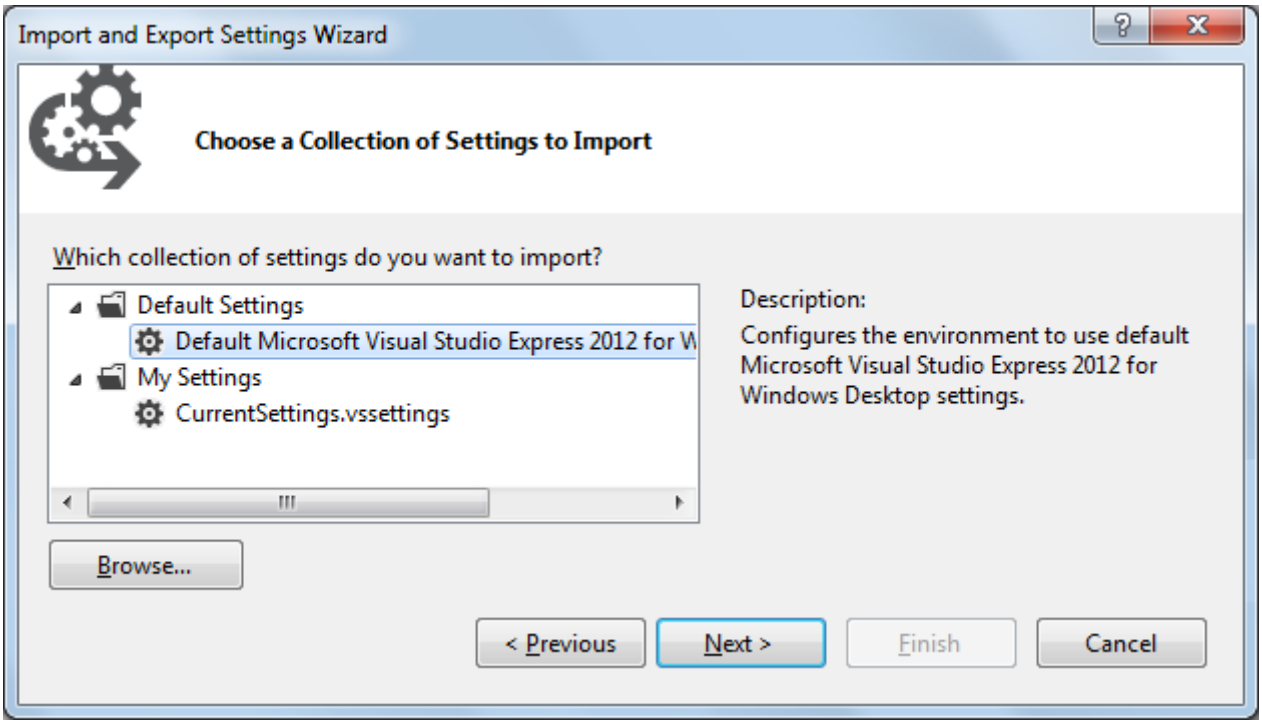

Рис. 2. Окно выбора коллекции настроек для применения

1.3 Обращение к справочной системе MSDN*

Для вызова справочной системы необходимо выбрать команду **View Help** в выпадающем меню **HELP** или отметить элемент в исходном тексте или на форме и нажать клавишу **F1**. Просмотр справки возможен либо в браузере, либо в окне **Help Viewer**, что настраивается с помощью пунктов меню **Set Help Reference / Launch in Browser** и **Set Help Reference / Launch in Help Viewer** выпадающего меню **HELP**. В процессе выполнения лабораторных работ предполагается использование среды **Help Viewer**. **Help Viewer** позволяет просматривать справку по разделам (вкладка **Contents**), используя поиск по ключевым словам (вкладка **Index**) либо поиск в заголовках справочных статей (вкладка **Search**).

1.4 Меню и команды интегрированной среды Visual Studio Express 2012

Чтобы выполнить команду в среде **Visual Studio**, можно воспользоваться тремя основными способами:

---

* **Microsoft Developer Network** (**MSDN**) – подразделение компании **Microsoft**, которое отвечает за взаимодействие с разработчиками. Однако при упоминании **MSDN** чаще всего имеют в виду **MSDN Library** – обширную библиотеку документов по технологиям **Microsoft**, либо **MSDN Subscription** – платную подписку на продукты **Microsoft**, включая **MSDN Library**. В лабораторном практикуме **MSDN ≡ MSDN Library**.

– с помощью главного меню,

– с помощью панелей инструментов (их видимость настраивается в меню **VIEW / Toolbars**),

– с помощью контекстного меню, которое активизируется правой кнопкой мыши.

1.4.1 Меню **FILE**

Команды выпадающего меню **FILE** можно использовать для работы как с проектами, так и с файлами исходного кода. Описание основных пунктов меню приведено в таблице (для получения более подробной информации здесь и далее следует обращаться к справочной системе или литературе):

New Project…	Создание нового проекта/решения
New Team Project…	Создание нового командного проекта/решения с использованием **Team Foundation Server**
Open Project…	Открытие проекта/решения
Open File…	Открытие файла
Close	Закрытие текущего файла
Close Solution	Закрытие решения
Save, **Save As**, **Save All**	Сохранение, соответственно, текущего файла, файла под другим именем или всего решения
Page Setup…, **Print…**	Настройка параметров страницы для печати исходного текста и, соответственно, печать
Recent Files, **Recent Projects and Solutions**	Выбор и открытие недавно открываемых файлов либо, соответственно, проектов и решений
Exit	Закрытие среды **Visual Studio**. Предварительно вызывается диалог сохранения изменений

1.4.2 Меню **EDIT**

Команды меню **EDIT** предназначены для отмены или повторения последнего действия (например, отмены ввода текста или добавления элемента управления на форму) – команды **Undo** и **Redo,** копирования (**Copy**), вырезания в буфер обмена (**Cut**)**,** удаления (**Delete**), вставки (**Paste**). Эти команды применимы как к выделенному текстовому блоку в окне редактора, так и к элементам управления, помещённым на форму.

Visual Studio обладает стандартными возможностями поиска и замены, сгруппированными в подменю **Find And Replace**. Поиск может производиться в тексте текущего документа (**Current Document**), во всех открытых документах (**All Open Documents**), во всех файлах проекта (**Current Project**) и т.д., что указывается в поле диалогового окна поиска **Look in** (рис. 3). Для команды **Quick Find** при обнаружении совпадения осуществляется переход на найденную строку с выделением искомого текста. Команда **Find in Files** выводит результаты поиска в окно **Find Results**. Соответственно, команда **Quick Replace** производит замены указанного фрагмента текста на другой. Команда **Replace in Files** выводит результаты замен в окно **Find Results**. Впрочем, различия **Quick**- и **in Files**-версий инструментов поиска и замены связаны в основном с компактностью диалогового окна.

Команда **Navigate to…** предназначена для поиска объектов: пространств имен (**namespace**), классов (**class**), структур (**struct**), интерфейсов (**interface**), типов (**type**), перечислений (**enum**) и т.д. Он позволяет находить в коде вхождения, используя возможности «нечеткого» поиска. Например, если написать **Form Calc** вместо **FormCalc**, **Navigate to…** всё равно выдаст верные результаты.

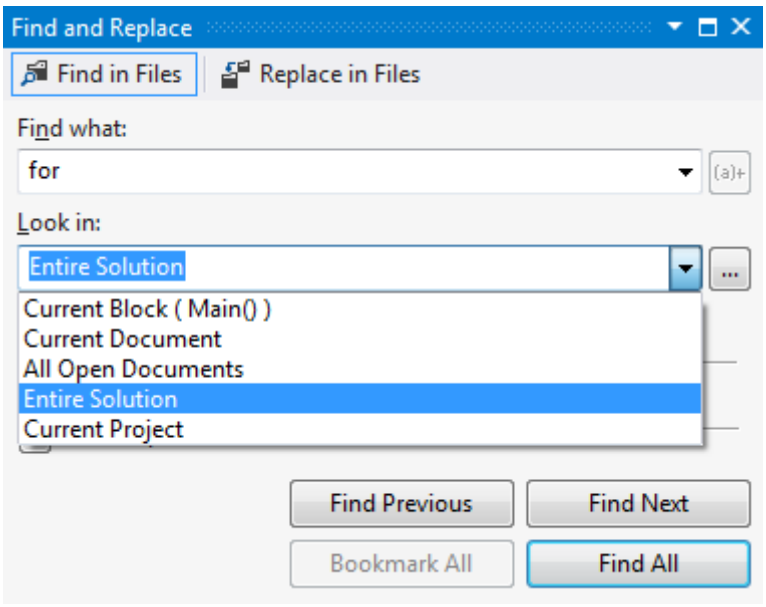

Рис. 3. Окно поиска **Find in Files**

С помощью пункта меню **Insert File As Text...** можно выполнить вставку в текущую позицию в редакторе содержимого выбранного файла.

Пункты подменю **Advanced** предоставляют дополнительные возможности по редактированию программного текста:

Format Document	Форматирование программного текста в активном окне с использованием стандартных отступов
Format Selection	Форматирование выделенного фрагмента программного текста с использованием стандартных отступов
View White Space	Делает видимыми символы пробела и табуляции
Word Wrap	Удаление горизонтальной полосы прокрутки
Incremental Search	Выполнение интерактивного поиска, позволяющего искать по части слова и уточнять полученные результаты в реальном времени. Занимает очень малую часть экрана и хорошо подходит для быстрого поиска в текущем файле
Comment Selection	Комментирование выделенных строк
Uncomment Selection	Раскомментирование выделенных строк

Пункты подменю **Bookmarks** предоставляют возможность работы с закладками для удобного перемещения по тексту программы, в том числе:

Toggle Bookmark	Создание закладки на текущей строке
Disable All Bookmarks	Выключение всех закладок на текущей строке
Enable Bookmark	Включение/выключение закладки на текущей строке
Previous Bookmark	Переход на предыдущую по тексту закладку
Next Bookmark	Переход на следующую по тексту закладку
Clear Bookmarks	Удаление всех закладок
Add Task List Shortcut	Добавление ярлыка списка задач. Содержимое выбранной строки кода в окне **Task List** (раздел **Shortcuts**) будет отображаться в качестве описания задачи. Двойной щелчок по такой задаче приведёт к переходу на эту строку кода в окне редактора

Пункты подменю **Outlining** позволяют работать со свойством скрытия/раскрытия содержимого фрагмента программы:

Toggle Outlining Expansion	Раскрытие/закрытие содержимого текущего фрагмента, на котором установлен курсор
Toggle All Outlining	Раскрытие/закрытие содержимого всех фрагментов в тексте программы
Stop Outlining	Выключение возможности раскрытия/закрытия фрагментов программы
Stop Hiding Current	Выключение возможности раскрытия/закрытия фрагментов программы в выделенной области
Collapse Do Definition	Включение возможности раскрытия/закрытия фрагментов программы

Возможности технологии **IntelliSence** доступны не только автоматически, но и через подменю **IntelliSence**, в том числе:

List Members	Отображение списка членов данного объекта. Для использования этой возможности в редакторе кода нужно ввести имя объекта, поставить точку и выбрать этот пункт меню
Parameter Info	Отображение подсказки о параметрах метода. Чтобы использовать эту возможность, курсор в редакторе устанавливается на вызов метода и вызывается данный пункт меню. С помощью стрелок ↑ и ↓ можно посмотреть различные варианты перегрузок метода
Quick Info	Вывод подсказки с информацией об объекте, внутри которого в редакторе кода установлен курсор редактирования
CompleteWord	Завершение текущего неполного слова в редакторе либо вывод списка возможных вариантов завершения
ToggleCompletionMode	Установление стандартного или расширенного режима завершения
Insert Snippet...	Вставка сниппета (готового фрагмента кода) в текст программы. Некоторое количество готовых сниппетов поставляется вместе с **Visual Studio**. Можно также добавлять свои, используя **Code Snippet Manager** (**Tools** / **Code Snippet Manager...**)
SurroundWith...	Создание для выделенного программного кода блока определённого вида (**if**, **for** …)

Пункт **Refactor** позволяет выполнить рефакторинг программного кода, т.е. изменение внутренней структуры программы, не затрагивающее её внешнего поведения и имеющие целью облегчить понимание её работы. В **Visual Studio Express 2012** доступны две возможности:

Rename…	Переименование идентификаторов для символов кода, таких как поля, локальные переменные, методы, пространства имен, свойства и т.п. Вносимые изменения можно предварительно просмотреть
Extract Method...	Создание нового метода на основе выделенного фрагмента кода

1.4.3 Меню **VIEW**

Позволяет переключаться в режим редактирования кода (команда **Code**) или на вкладку дизайнера форм (**Designer**). Следующие группы команд управляют отображением окон среды **Visual Studio**, в частности, позволяют отобразить обозреватель решений (**Solution Explorer**), иерархию вызовов (**Call Hierarchy**), палитру компонентов (**Toolbox**) и окно свойств компонента (**Properties Window**).

При вызове **Call Hierarchy** для метода, свойства, индексатора или конструктора в исходном коде будет показано дерево всех входящих и исходящих вызовов этого метода, свойства и т.д. Каждый метод, свойство или конструктор в узлах **Calls To** и **Calls From** можно развернуть в его собственные узлы **Calls To** и **Calls From**. **Call Hierarchy** хорошо подходит для навигации по коду и для понимания связей между методами.

1.4.4 Меню **PROJECT**

Команды меню **PROJECT,** начинающиеся с приставки **Add**, позволяют добавить в проект новый элемент – форму, элемент пользовательского интерфейса, компонент, класс, файл с исходным текстом и т.д. Команда **Exclude From Project** позволяет исключить элемент из проекта. Команда **Add Reference…** открывает окно **Reference Manager** (рис. 4) и позволяет подключить, при необходимости, дополнительные сборки (библиотеки). В разделе **Framework** располагаются все сборки платформы **.NET**, установленные на компьютере, **Extensions** – все сторонние сборки. В разделе **COM** находятся библиотеки **COM**, установленные на компьютере. Раздел **Solution** позволяет добавить ссылки на другие проекты текущего решения, **Browse** содержит список недавно добавляемых (**Recent**) сборок. Уже имеющиеся ссылки можно увидеть в списке **References** окна **Solution Explorer**.

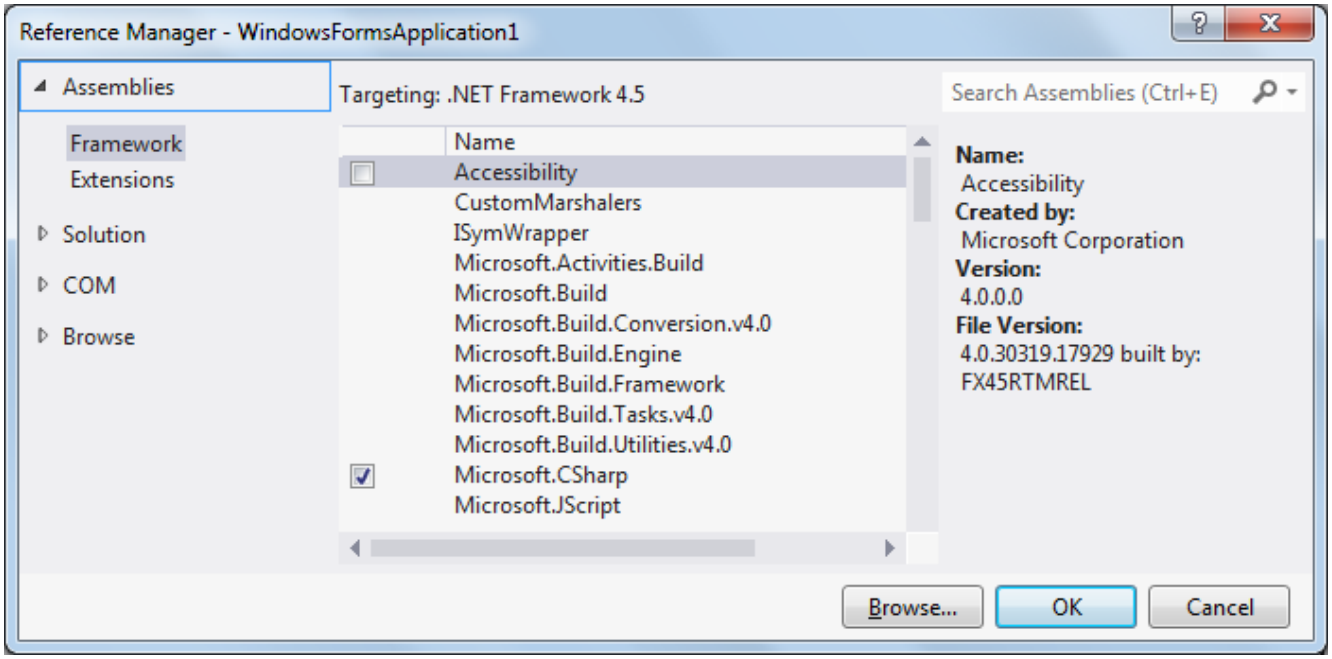

Рис. 4. Окно **Reference Manager**

Set as SrartUp Project позволяет назначить текущий проект как запускаемый по умолчанию, если у в решении находится несколько проектов.

Редактирование свойств проекта позволяет производить подменю **Properties** (название команды начинается с имени проекта, например **WindowsFormsApplication1 Properties**).

1.4.5 Меню **BUILD**

Основные команды меню **BUILD**:

Build Solution	Сборка решения (компиляция всех входящих в решение проектов и сборка исполняемого файла либо файла динамической библиотеки)
Rebuild Solution	Повторная сборка решения

Clean Solution	Удаление всех результатов сборки решения

1.4.6 Меню **DEBUG**

Основные команды меню **DEBUG**:

Windows	Команды этого подменю управляют отображением окон **Output**, **Breakpoint** и **Immediate**
Start Debugging	Начинает выполнение программы с отображением отладочной информации
Start Without Debugging	Выполнение программы без отображения отладочной информации, все точки прерывания (**breakpoints**) игнорируются
Step Into	Выполнение пошаговой трассировки программы с заходом во все функции
Step Over	Выполнение трассировки без захода в функции
Toggle Breakpoint	Установление или удаление точки прерывания в текущей строке программы

1.4.7 Меню **TOOLS**

Меню содержит различные внешние инструментальные средства, выбор компонентов окна **Toolbox** (**Choose Toolbox Items…**), настройки панелей инструментов и меню (**Customize…**), настройки параметров **Visual Studio** (**Options…**).

1.4.8 Меню **WINDOW**

Предназначено для управления окнами редактора **Visual Studio**. Позволяет, в частности, дублировать текущий файл в нескольких окнах (**New Window** или **Split**), разделять окно редактора (**New Horizontal Tab Group**, **New Vertical Tab Group**), что позволяет открывать одновременно несколько файлов, закреплять вкладки в левой части редактора (**Pin Tab**), производить переключение между вкладками редактора. Команда **Reset Window Layout** возвращает расположение окон среды в исходное состояние.

1.5 Панели инструментов и палитра компонентов. Окно свойств

Панели инструментов скрываются или отображаются с помощью меню **VIEW / Toolbars**. Настройка панелей инструментов осуществляется с помощью команды **VIEW / Toolbars / Customize…**.

Окно **Toolbox** представляет собой палитру компонентов, распределённых по категориями: **All Windows Forms** – общий список компонентов, **Common Controls** – элементы управления общего назначения и т.д. Для добавления на форму элемента управления существует несколько путей:

– щёлкнуть мышью на нужном элементе управления в окне **Toolbox**, затем растянуть на форме рамку, которая и определит его размеры;

– дважды щёлкнуть на элементе управления в окне **Toolbox**, и он будет добавлен на форму, причем размеры и положение **Visual Studio** определит автоматически (по умолчанию);

– щёлкнуть мышью на нужном элементе управления, затем один раз щёлкнуть на форме; элемент будет добавлен так, чтобы его верхний левый угол совпадал с координатами курсора мыши, а размер будет задан по умолчанию (если всё время держать нажатой клавишу **Ctrl**, будут добавлено несколько экземпляров этого элемента управления – при каждом щелчке мыши).

Невизуальные компоненты добавляются аналогично, но вне зависимости от указанного места всегда располагаются в специальной области по формой (панели компонентов).

Окно **Properties** служит для установки свойств компонента. На вкладке **Properties** редактируются свойства компонентов, на вкладке **Event** (рис. 5) задаются обработчики событий.

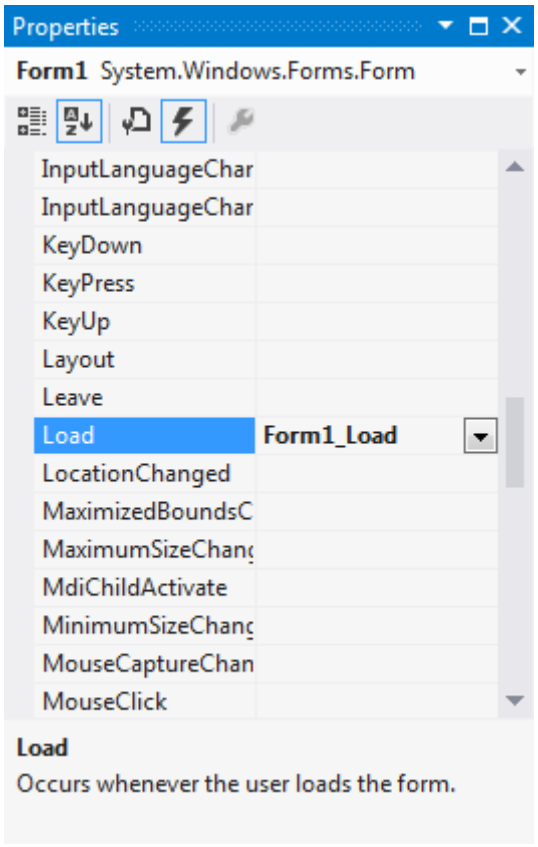

Рис. 5. Окно **Properties** (вкладка **Events**)

В окне **Properties** отображены не все свойства компонентов, а лишь те, которые могут быть установлены на этапе проектирования. Все изменения в окне **Properties** вызывают изменения исходного кода программы (для формы **Form1** – файла **Form1.Designer.cs**).

1.6 Контекстные меню

Не все команды **Visual Studio** доступны через выпадающие меню. Иногда для некоторых окон или областей окна приходится (и даже удобнее) использовать контекстные меню. Например, с помощью контекстного меню ярлычка открытого файла в редакторе можно сохранить этот файл (**Save**), закрыть его (**Close**), закрыть все остальные файлы, кроме данного (**Close All But This**), скопировать в буфер обмена полный путь к файлу (**Copy Full Path**), открыть в проводнике каталог, в котором записан данный файл (**Open Containing Folder**).

Пункт контекстного меню редактора кода **Go To Definition** переносит к определению заданного символа и работает с методами, типами, классами, членами и т.д. Может быть полезен при исследования деталей реализации, особенно при чтении кода.

2. Приложения Windows Forms

2.1 Проекты и решения. Файлы, создаваемые средой

Сеанс работы в **Visual Studio Express 2012 for Windows Desktop** начинается с открытия существующего или создания нового решения. Решение может включать один или нескольких проектов. Для программы на **C#** всегда имеется файл проекта (**.cspro**) и файл решения (**.sln**). Проект как часть решения состоит из отдельных элементов, например, файлов, описывающих

форму, файлов с исходными текстами (**.cs**), **XML**- и **HTML**-файлов, графических файлов и т.д. Концепция решений помогает объединить проекты и другие элементы в одном рабочем пространстве. Открытие в **Visual Studio** файла проекта эквивалентно открытию файла всего решения.

Решение сохраняется в каталоге с произвольным именем, который содержит файл решения (**.sln**), включающий данные обо всех проектах, входящих в решение. Для каждого проекта отводится при сохранении отдельный каталог.

Для проекта **Visual Studio** создаёт ряд файлов. Наиболее важными являются следующие:

– основной файл проекта (.**csproj**), который, кроме всего прочего, перечисляет имена других файлов, составляющих проект; имеется только один **csproj**-файл для каждого проекта;

– файл **Program.cs**, который содержит описание класса **Program** с методом **Main()**, создающим и открывающим форму (экземпляр класса **Form**) для интерактивной работы пользователя;

– файлы с исходным кодом (**.cs**).

В подкаталоге проекта **bin**/**Debug** находится исполняемый **exe**-файл проекта. Для его запуска не нужен **Visual Studio**, но должен быть установлен **.NET Framework** соответствующей версии.

2.2 Проектирование форм

Проектирование форм – ядро визуальной разработки по технологии **Windows Forms.** Каждый помещаемый на форму элемент управления или любое задаваемое свойство вносит изменения в исходный код файла, связанного с формой.

При работе с формой на этапе проектирования можно изменять её свойства, свойства одного или нескольких элементов управления одновременно. Чтобы выбрать форму или элемент управления, можно просто щёлкнуть по нему мышью. Возможен выбор нескольких элементов управления – с помощью растягивающейся рамки либо с помощью комбинации клавиш **Ctrl** + щелчок левой кнопкой мыши. Для изменения относительного расположения перекрывающихся элементов управления можно использовать команды контекстного меню **Bring to Front** и **Send to Back**.

Для управления взаимным расположением элементов управления используется пункт главного меню **FORMAT**. Основные команды:

Align	Управление выравниванием элементов управления относительно одного из них (первого выделенного). Выравнивание происходит по мнимой линии, проходящей через прямоугольную границу одного из элементов управления: левую (**Lefts**), правую (**Rights**), верхнюю (**Tops**), нижнюю (**Bottoms**) или по центру, вертикально или горизонтально (**Middles**)
Make Same Size	Выравнивание размеров элементов управления – по ширине (**Width**), высоте (**Height**) или и по ширине и по высоте (**Both**)
Horizontal Spacing, Vertical Spacing	Управление расстоянием между элементами управления по вертикали или горизонтали
Center in Form	Расположение элементов управления в центра формы по вертикали или горизонтали. При выборе нескольких элементов они будут размещены так, чтобы центр мнимого прямоугольника, границы которого совпадают с границами крайних компонентов, оказался в центре
Lock Controls	Привязывает выделенные элементы управления к их текущей позиции, делая невозможным перемещение

Команда меню **VIEW / Tab Order** позволяет настроить порядок обхода элементов управления при их выделении во время выполнения программы с помощью клавиши **Tab**.

2.3 Написание программного кода

Создание приложений **Windows Forms** обычно заключается в написании обработчиков событий. Когда событие происходит (например, щелчок мышью на кнопке), **Windows** посылает приложению сообщение, информируя его об этом событии. Реакция приложения состоит в получении сообщения о событии и вызове соответствующего метода отклика. Для каждого вида компонентов определён ряд различных событий. О событиях, доступных для компонента, можно узнать при рассмотрении страницы **Events** окна **Properties**.

Дизайнер форм в **Visual Studio** работает полностью на основе исходного кода приложения, и любые изменения сгенерированного им кода приводят к изменениям и в окне дизайнера. При создании нового **Windows**-приложения будет сгенерирован следующий сходный код (файл **Form1.cs**):

```
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;

namespace WindowsFormsApplication1
{
    public partial class Form1 : Form
    {
        public Form1()
        {
            InitializeComponent();
        }
    }
}
```

Слова, которые подсвечиваются синим цветом, являются ключевыми (зарезервированы). Ключевое слово **using** означает использование пространства имён. Все идентификаторы, объявленные в пространстве имён, становятся доступны в проекте. Пространство имён **System** содержит базовые классы, которые определяют часто используемые типы данных, события (**events**) и обработчики событий (**event handlers**), интерфейсы (**interfaces**), атрибуты (**attributes**) и исключения (**exceptions**). Ключевое слово **namespace** означает указание пространства имён, в котором будут объявлены идентификаторы. Все идентификаторы должны быть объявлены во всех файла проекта (решения) в одном пространстве имен, чтобы можно было получить к ним доступ из методов любого класса. **WindowsFormsApplication1** – имя проекта по умолчанию, объявленное при создании проекта. Наибольший интерес представляет строка

```
public partial class Form1 : Form
```

В данной строке обявляется класс **Form1**, наследуемый от класса **Form**. Этот класс является классом формы, которую можно увидеть в дизайнере форм. Оператор «:» означает наследование от базового класса. Ключевое слово **public** является модификатором уровня доступа и указывает на отсутствие каких-либо ограничений на уровень доступа к классу и его полям. Ключевое слово **partial** указывает, что класс описывается (частями) в нескольких фай-

лах. Метод **public Form1()** является конструктором класса. Именно этот метод вызывается при создании нового экзамепляра класса. Метод класса **Form1 InitializeComponent()** описан в другом файле – **Form1.Designer.cs:**

```
private void InitializeComponent()
{
     this.components = new System.ComponentModel.Container();
     this.AutoScaleMode = System.Windows.Forms.AutoScaleMode.Font;
     this.Text = "Form1";
}
```

Ключевое слово **this** используется для обращения к текущему экзмепляру класса, так как заранее неизвестно, какое имя будет иметь переменная типа **Form1**. Свойство **Text** – текст, отображаемый в заголовке окна.

Если разместить на форме кнопку (**Button**), то можно будет наблюдать следующие измнения в исходном коде:

```
private void InitializeComponent()
{
     this.button1 = new System.Windows.Forms.Button();
     this.SuspendLayout();
     //
     // button1
     //
     this.button1.Location = new System.Drawing.Point(205, 33);
     this.button1.Name = "button1";
     this.button1.Size = new System.Drawing.Size(75, 23);
     this.button1.TabIndex = 0;
     this.button1.Text = "button1";
     this.button1.UseVisualStyleBackColor = true;
     //
     // Form1
     //
     this.AutoScaleDimensions = new System.Drawing.SizeF(6F, 13F);
     this.AutoScaleMode = System.Windows.Forms.AutoScaleMode.Font;
     this.ClientSize = new System.Drawing.Size(292, 273);
     this.Controls.Add(this.button1);
     this.Name = "Form1";
     this.Text = "Form1";
     this.ResumeLayout(false);
}
```

Метод **SuspendLayout()** отключает механизм автоматического размещения элементов управления на форме и позволяет указывать их координаты и размеры вручную. Свойства **AutoScaleDimensions** и **AutoScaleMode** устанавливают различные параметры автоматического масштабирования. Необходимости в такой операции возникает из-за того, что мониторы имеют различные физические размеры и различное разрешение экрана. Более подробную информацию по данному вопросу можно получить в соотвествующем разделе справки. Стоит отметить, что в строке

```
this.AutoScaleDimensions = new System.Drawing.SizeF(6F, 13F);
```

ключевое слово **new** означает создание нового экземпляра класса. **System.Drawing.SizeF** в данном контексте означает вызов конструктора, **6F** и **13F** – параметры (**F** означает, что тип данных – **float**, и присутствует в тексте для того, чтобы избежать лишнего преобразования типов). Поле **ClientSize** формы определяет размер её клиентской области, без рамки и заголовка окна. **Name** – внутреннее имя формы, которое может быть использовано, к примеру, для получения ссылки на экземпляр класса по его имени.

Вызов **this.ResumeLayout(false)** включает механизм автоматического размещения элементов управления, при этом их незамедлительное автоматическое размещение вызвано не будет, об этом говорит параметр **false**.

В третьей строке создается кнопка, объявленная в том же классе **Form1** как

```
private System.Windows.Forms.Button button1;
```

Строки, начинающиеся с **this.button1**, устанавливают определённые свойства кнопки. Все свойства, которые не были инициализированы явно, инициализируются в момент создания кнопки в её конструкторе. В строке **this.Controls.Add(this.button1)** происходит добавление кнопки в коллекцию **Controls** формы, после этого кнопка будет отображаться на форме.

Установить для кнопки обрабочик события можно двумя способами: с помощью списка событий окна **Properties,** для этого нужно дважды щёлкнуть левой кнопкой мыши в поле рядом с именем события – тогда будет автоматически сгенерирован метод, который будет вызываться при возникновении события. Кроме того, двойной щелчок по кнопке создаёт метод обработки события нажатия на кнопку **Click**:

```
private void button1_Click(object sender, EventArgs e)
{

}
```

При этом в методе **InitializeComponent**() появится строка:

```
this.button1.Click += new System.EventHandler(this.button1_Click);
```

Операция += вместо простого присваивания используется ввиду того, что одному событию может быть сопоставлено несколько обработчиков.

2.4 Отладка приложения

Чтобы запустить программу на выполнение в отладчике, используется команда **DEBUG / Start Debugging** (клавиша **F5**). Выполнение программы при запуске в отладчике будет приостановлено при достижении точки прерывания. После приостановки можно перейти к пошаговому выполнению программы, используя команды **Step Into** или **Step Over** меню **DEBUG**, либо продолжить обычное выполнение программы, выбрав **DEBUG / Continue**.

Слежение за значением идентификаторов можно производить с помощью окна **Watch** (рис. 6), используя команду **Add Watch** контекстного меню.

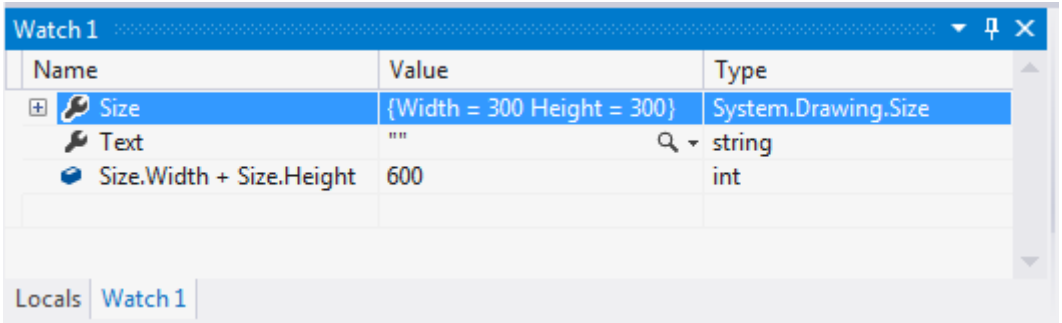

Рис. 6. Окно **Watch**

Чтобы отслеживать значение, нужно сначала добавить в окно выражение, соответствующее этому значению (к примеру, можно отслеживать не значение одной переменной, а сумму

двух переменных). На каждом шаге идентификаторы, изменившие свои значения, в окне **Watch** будут подсвечиваться красным цветом.

Для добавления выражений в окно **Watch** и быстрого просмотра их значений можно использовать команду **DEBUG** / **QuickWatch**, после выбора которой появляется окно **QuickWatch** (рис. 7).

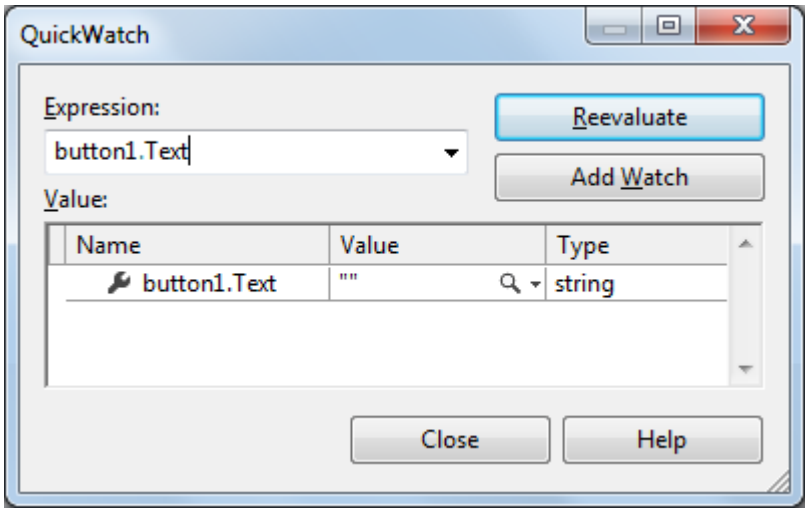

Рис. 7. Окно **QuickWatch**

Значение (или значения), хранящиеся в объекте, также можно узнать при наведении курсора мыши на его идентификатор.

3. Типы данных языка C#

3.1 Обзор типов данных

Система встроенных типов языка **C#** содержит практически все встроенные типы кроме **long double**) языка **C++**, но при этом уточняет и дополняет его. В частности, тип **string** является встроенным в язык. При этом, в отличие от **C++**, в языке **C#** все типы являются объектами.

Все типы языка **C#** можно разделить на следующие категории:

1. Типы-значения, или *значимые* типы.
2. Ссылочные типы.
3. Указатели.
4. Тип **void**.
5. Тип **object**.

Эта классификация основана на том, где и как хранятся значения типов. Для значимого типа используется прямая адресация, значение хранит собственно данные, и память для него отводится, как правило, в стеке. Для ссылочного типа значение задает ссылку на область памяти в «куче», где расположен соответствующий объект. Указатели имеют ограниченную область действия и могут использоваться только в небезопасных блоках, помеченных как **unsafe**. Особый статус имеет и тип **void**, указывающий на отсутствие какого-либо значения.

Все встроенные типы языка **C#** однозначно отображаются на системные типы **.NET Framework**, размещёнными в пространстве имён **System**. Поэтому всюду, где можно использовать имя типа, например, **int**, точно так же можно использовать и имя **System.Int32.**

Базовым для всех типов является тип **object** (**System.Object**). Поэтому в «объектной» переменной может быть сохранёно значение любого другого типа.

В **C# 3.0** появилась возможность определения переменных без явного указания их типов. Компилятор получил возможность выводить тип данных на основе кода. Этот процесс называется *выведением локального типа,* или *неявным типизированием*. Описание переменной

```
var i = 8;
```

является полностью эквивалентным традиционному описанию с явным указанием типа:

```
System.Int32 i = 8;
```

Неявное типизирование не является возвратом к универсальному типу данных (такому как **object**). На самом деле компилятор строго типизирует переменную, выбирая тип данных в зависимости от кода. При этом переменной для типизации компилятором должно быть присвоено значение. Кроме того, выведение типа работает только с локальными типами. Оно не работает с переменными уровня класса (полями) или статическими переменными.

3.2 Типы, содержащие значение

3.2.1 Целочисленные типы

Таблица 1. Типы данных языка **C#**

Тип	Тип .NET Framework	Область значений	Размер
sbyte	**System.SByte**	-128 .. 127	8-бит знаковый
byte	**System.Byte**	0 .. 255	8-бит беззнаковый
char	**System.Char**	U0000 .. Uffff	16-бит **Unicode** символ
short	**System.Int16**	-32 768 .. 32 767	16-бит знаковый
ushort	**System.UInt16**	0 .. 65 535	16-бит беззнаковый
int	**System.Int32**	-2 147 483 648 .. 2 147 483 647	32-бит знаковый
uint	**System.UInt32**	0 .. 4 294 967 295	32-бит беззнаковый
long	**System.Int64**	-9 223 372 036 854 775 808 .. 9 223 372 036 854 775 807	64-бит знаковый
ulong	**System.UInt64**	0 .. 18 446 744 073 709 551 615	64-бит беззнаковый

3.2.2 Вещественные типы

Таблица 2. Вещественные типы языка **C#**

Тип	Тип .NET Framework	Аппроксимированный диапазон	Размер	Точность
float	**System.Single**	±1.5e-45 .. ±3.4e38	4 байта	7 разрядов
double	**System.Double**	±5.0e-324 .. ±1.7e308	8 байт	16 разрядов

В табл. 2 помимо диапазона указана точность (**precision**) в числе знаков после запятой.

3.2.3 Десятичный тип

Таблица 3. Десятичный тип языка **C#**

Тип	Тип .NET Framework	Аппроксимированный диапазон	Размер	Точность
decimal	**System.Decimal**	±1.0×10e-28 .. ±7.9×10e28	12 байт	28 разрядов

Тип **decimal** по сравнению с вещественными типами имеет высокий порядок точности и меньший диапазон, что делает его удобным при финансовых расчётах.

3.2.4 Логический тип

Таблица 4. Логический тип языка **C#**

Тип	Тип .NET Framework	Значения	Размер
bool	**System.Boolean**	**true**, **false**	1 байт

Тип **bool**, как и в **C++**, используется для объявления логических переменных, принимающих значения «ложь» (**false**) и «истина» (**true**). Но в отличие от **C++**, в **C#** нет соответствия между логическими и целочисленными значениями.

3.2.5 Структуры

Структуры обычно используются для объединения небольшого числа связанных переменных, например:

```
public struct Book
{
    public string author;
    public string title;
}
```

3.2.6 Перечисления

Перечисления – особый тип, содержащий множество буквенных констант. Константа определяется числом (по умолчанию нумерация начинается с нуля), и если значение не указано явно, то следующей константе будет присвоен следующий номер, увеличенный на 1. Например:

```
enum Colors {Red = 1, Green, Blue = 4, Yellow = 8};
```

3.3 Ссылочные типы

3.3.1 Строки

Тип **string** (**System.String**) описывает объект, представляющий собой коллекцию объектов **System.Char** – **Unicode**-символов, используемых для отображения текста. Над строками определены следующие операции:

- присваивание (=),
- две операции проверки эквивалентности (==) и (!=),
- конкатенация или сцепление строк (+),
- взятие индекса ([]).

Строки в **C#** являются полноценным ссылочным типом, но при этом поддерживают операцию сравнения, характерную для типов-значений (строки равны, если равны их значения).

Возможность взятия индекса означает, что строку можно рассматривать как массив и обращаться отдельно к каждому её символу. Символ строки доступен только для чтения: строка не может быть изменена после своего создания. Методы, используемые для модификации строки, на самом деле создают новый объект **string**. Если же необходимо постоянное изменение содержимого, то лучше использовать класс **System.Text.StringBuilder** (подробнее см. п. 3.3.2).

Операции, разрешенные над строками в **C#**, разнообразны. Методы этого класса (табл. 5) позволяют выполнять вставку, удаление, замену, поиск вхождения подстроки в строку.

Таблица 5. Основные методы класса **String**

Метод	Описание
IndexOf(), **LastIndexOf**(), **IndexOfAny**(), **LastIndexOfAny**()	Возвращает индекс первого (последнего) вхождения заданной подстроки (или любого символа из заданного набора символов)
Insert()	Вставляет указанный экземпляр **String** в данный экземпляр по заданному индексу
PadLeft(), **PadRight**()	Выравнивает символы в данном экземпляре строки по левому (правому) краю, добавляя справа (слева) пробелы или указанные символы до указанной общей длины
Remove()	Удаляет подстроку из заданной позиции
Replace()	Заменяет подстроку в заданной позиции на новую подстроку
Split()	Возвращает строковый массив, содержащий подстроки данного экземпляра **String**, разделенные элементами заданной строки или массива символов
StartsWith(), **EndsWith**()	Возвращает **true** или **false**, в зависимости от того, начинается (заканчивается) ли строка заданной подстрокой
Substring()	Извлекает подстроку из данного экземпляра
ToCharArray()	Копирует символы данного экземпляра строки в массив символов
ToLower(), **ToUpper**()	Возвращает копию этого объекта **String**, переведенную в нижний (верхний) регистр
Trim(), **TrimStart**(), **TrimEnd**()	Удаляет все начальные и/или конечные вхождения заданных наборов символов из текущего объекта **String**

Можно рассмотреть применение этих методов на одном примере.

```
// Задание исходной строки:
string str = "Технологии программирования-2023";
// Нахождение позиции в строке, с которой начинается подстрока "2023":
int indSep = str.IndexOf("2023");
// Задание вспомогательного массива символов и его инициализация:
char[] chArray = new char[] {' ','-'};
// Нахождение позиции в строке любого из символов, заданных в массиве chArray:
```

```
int indFiveSep = str.IndexOfAny(chArray);
// Вставка в строку перед пробелом ещё двух пробелов:
str = str.Insert(indFiveSep, "  ");
// Дополнение строки справа двумя восклицательными знаками:
str = str.PadRight(str.Length + 2, '!');
// Удаление из строки двух символов (повторный поиск позиции методом IndexOf()
// требуется из-за модификации строки), т.е. подстроки "20":
str = str.Remove(str.IndexOf("2023"), 2);
// Поиск и замена в строке подстроки "23" на "2023":
str = str.Replace("23", "2023");
// Получение массива из строк "Технологии", "программирования" и "2023"
// (используется массив из трёх разделителей и опция подавления пустых элементов
// в результирующем массиве spArray):
string[] spArray = str.Split(new char[] {' ','-','!'},
     StringSplitOptions.RemoveEmptyEntries);
 // Проверка, начинается ли строка в ячейке массива с номером 1 с подстроки "про":
bool ask = spArray[1].StartsWith("про");
// Извлечение из строки подстроки длиной 5 символов, начиная с позиции 0:
string strSub = str.Substring(0, 5);
// Преобразование строки в массив символов:
char[] chStrSub = strSub.ToCharArray();
// Преобразование всех символов строки в нижний регистр:
strSub = strSub.ToLower();
// Добавление в начало строки трёх пробелов (для демонстрации Trim-метода):
strSub = "   " + strSub;
// Получение строки без начальных пробелов:
strSub = strSub.TrimStart();
```

В приложениях **Windows Forms** обычно строковое значение вводится в текстовое поле, поэтому задать значение переменной str можно было бы следующим образом:

```
string str = textBoxMy.Text;
```

Строка может содержать служебные символы типа **\n** или **\t**, которые начинаются с наклонной черты (\) и используются в данном случае для указания перевода строки или вставки символа табуляции. Поскольку наклонная черта влево самостоятельно используется в некоторых синтаксисах строк, типа URL или путей на локальном диске, то в такой строке наклонной черте должен предшествовать другой символ наклонной черты. Строки могут также быть созданы с помощью «дословной» записи строки. Такие строки должны начинаться с символа (@), который сообщает конструктору, что строка должна использоваться дословно, даже если она включает служебные символы. Следующие два определения эквивалентны:

```
string stringOne = "C:\\MyDirectory\\MyFile.txt";
string stringTwo = @"C:\MyDirectory\MyFile.txt";
```

3.3.2 Класс **StringBuilder**

Строковый класс **StringBuilder**, определённый в пространстве имён **Svstem.Text**, позволяет напрямую изменять свои существующие объекты. Класс **StringBuilder** динамически перераспределяет память, используемую строкой. Как только размер строки превышает размер буфера, объём буфера вырастает вдвое. Для преобразования объекта **StringBuilder** в объект **String** служит метод **ToString**().

```
StringBuilder strBuild = new StringBuilder("Марс");
strBuild[0] = 'Б';
string strSimple = strBuild.ToString();
```

Над строками этого класса определены практически те же операции с той же семантикой, что и над строками класса **String**:

- присваивание (=);
- две операции проверки эквивалентности (==) и (!=);
- взятие индекса ([]).

Операция конкатенации (+) не определена, её роль играет метод **Append**(), дописывающий новую строку в конец уже существующей.

3.3.3 Массивы

Массив в языке **C#** – это тип, производный от класса **System**.**Array**, поэтому все массивы обладают общим набором членов. В **C#**, как и в **C++**, нумерация элементов массива начинается с 0. Массивы могут быть простыми и многомерными. Они объявляются путём помещения квадратных скобок после указания типа данных для элементов этого массива. Например:

```
int[] masInt; // определение целочисленного массива
masInt = new int[5]; // создание массива из 5 элементов
```

Число элементов массива можно узнать, воспользовавшись его свойством **Length**.

Объявление массива и его создание можно совместить:

```
int[] masInt = new int[5];
```

Обращение к ячейкам массива возможно только после его создания:

```
masInt[0] = 3; // запись значения в 1-ю ячейку массива
masInt[4] = 2; // запись значения в последнюю ячейку массива
```

Элементы массива можно задать сразу при объявлении:

```
string[] masStr = new string[2] {"Сатурн", "Юпитер"};
```

В приведённом примере `new string[2]` использовать не обязательно: размер массива определился бы по количеству перечисленных в фигурных скобках строк.

В отличие от массивов в **C++**, в **C#** элементам массива автоматически присваиваются значения по умолчанию в зависимости от используемого для них типа данных. Например, для массива целых чисел всем элементам будет изначально присвоено значение 0, для массива объектов – значение **null** и т.д.

Многомерный массив в **C#** определяется следующим образом:

```
int[,] mtrInt; // определение двумерного массива (матрицы) целых чисел
mtrInt = new int[2, 3]; // создание матрицы из 2-х строк и 3-х столбцов
```

Массив, приведённый в предыдущем примере, относится к числу прямоугольных. Возможны и «ступенчатые» массивы. Например:

```
int[][] smInt = new int[2][]; // определение двумерного ступенчатого массива
smInt[0] = new int[3]; // первая строка - массив из 3-х элементов
smInt[1] = new int[10]; // вторая строка - массив из 10 элементов
```

Для ступенчатого массива задаётся несколько пар квадратных скобок – по размерности массива. Точно так же они используются при записи в массив или чтении из него. Например:

```
smInt[1][9] = 12; // запись значения в послений элемент массива
```

Основные методы класса **Array** указаны в табл. 6.

Таблица 6. Основные методы класса **Array**

Метод	Описание
Clear()	Выполняет начальную инициализацию элементов массива. В зависимости от типа элементов устанавливает значение 0 для арифметического типа, **false** – для логического типа, "" – для строк
Copy()	Копирует часть или весь массив в другой массив
CopyTo()	Копирует все элементы одномерного массива в другой одномерный массив, начиная с заданного индекса
GetLength()	Возвращает количество элементов в заданном измерении массива
GetLowerBound(), **GetUpperBound**()	Возвращает нижнюю (верхнюю) границу по указанному измерению. Для массивов нижняя граница всегда равна нулю
IndexOf(), **LastIndexOf**()	Возвращает индекс первого (последнего) экземпляра значения в одномерном массиве или в части массива
Reverse()	Изменяет порядок элементов в одномерном массиве или в части массива на обратный

3.4 Получение сведений о типе

В классе **Object** – а значит, и у всех его потомков – определён метод **GetType**(), позволяющий узнать тип объекта-переменной. Метод **GetType**() возвращает тип **FCL**, т.е. тот тип, на который отражается тип языка и с которым реально идёт работа при выполнении программы. Например, в результате выполнения следующего программного кода

```
byte btVar = 0;
labelTypes.Text = btVar.GetType().ToString();
```

в надпись **labelTypes** будет занесена строка «**System.Byte**».

Функция **sizeof**() возвращает размер внутреннего представления заданного типа. Например, вызов **sizeof**(**int**) возвратит значение 4.

Чтобы узнать минимальное и максимальное значение типа данных, следует использовать свойства **MinValue** и **MaxValue** типа, например: **int**.**MinValue**.

3.5 Приведение и преобразование типов данных

Существует явное и неявное приведение типов. Неявное приведение возможно только при присваиваниях, в которых не может произойти потери данных, т.е. конечный тип должен содержать в себе все значения исходного типа (но для всех типов существует неявное преобразование к типу **object**). Обратное преобразование должно сопровождаться явным приведением:

```
int intVar = 10;
float floatVar = 0;
// Неявное приведение (данные не могут быть искажены):
```

```
floatVar = intVar;
intVar = 0.5F;
// Явное приведение (возможна потеря данных):
intVar = (int)floatVar;
```

Для ссылочных типов помимо приведения вида (<**Тип**>)<**Object**> можно использовать оператор **as** – <**Object**> **as** <**Тип**>:

```
(sender as Button).Text = "Надпись";
```

Перед приведением ссылочных типов, если тип заранее не известен, полезна проверка совместимости типов, выполняемая с помощью оператора **is**. Результат оператора **is** – логическое значение, которое можно использовать с условными операторами:

```
if (sender is Button)
... ... ...
```

Для типов, содержащих значения, часто необходима и более общая операция – преобразование (конвертация) типов, которая выполняется над «разнородными» типами. Для этого используется класс **Convert**, определенный в пространстве имен **System**. Методы класса **Convert** поддерживают общий способ выполнения преобразований между типами. Класс содержит 15 статических методов вида **To**<Тип> (**ToBoolean**(), ... **ToUInt64**()).

Важным видом преобразований являются преобразования в строковый тип и наоборот. Преобразования в строковый тип всегда определены, поскольку все типы являются потомками базового класса **Object**, а следовательно, обладают методом **ToString**().

Также все типы данных предоставляют возможность генерировать переменную этого типа, который задаётся в текстовом виде. Например:

```
// Текстовая строка указывается непосредственно:
bool boolVar = bool.Parse("True");
// Число должно быть введено в поле ввода класса TextBox:
int intVar = System.Int32.Parse(textBoxParamX.Text);
```

Впрочем, эта же функциональность может быть реализована средствами класса **Convert**:

```
bool boolVar = System.Convert.ToBoolean("True");
int intVar = Convert.ToInt32(textBoxParamX.Text);
```

4. Формы и элементы управления Windows Forms

4.1 Работа с несколькими формами

По умолчанию при запуске приложения в файле **Program.cs** автоматически выполняется создание экземпляра класса первой формы проекта. После добавления второй формы (например, можно назвать её **FormSecond**) в проект в одном из обработчиков событий основной формы можно создать её экземпляр и вывести на экран:

```
FormSecond formSecond = new FormSecond();
formSecond.Show();
```

Для закрытия (точнее, скрытия) формы используется метод **Hide**(). Реально методы **Show**() и **Hide**() всего лишь устанавливают свойство **Visible** формы.

Обращение к полям, свойствам и методам, а также элементам управления вспомогательной формы далее выполняется с помощью имени объектной переменной (в примере

formSecond). Но элементы управления по умолчанию имеют модификатор доступа **private**. Например, для элемента **Label**, размещёного на форме **FormSecond**, в файле **FormSecond.Designer.cs** будет сгенерировано объявление:

```
private System.Windows.Forms.Label labelMy;
```

Для доступа к labelMy по имени объектной переменной формы нужно заменить модификатор на **public** (этот способ нарушает инкапсуляцию и обычно не рекомендуется, но является самым простым).

4.2 Иерархия элементов управления. Класс Control

Каждый вид элементов управления описывается собственным классом. Довольно высокое положение в иерархии классов занимает класс **Control**. Он задаёт важные свойства, методы и события, наследуемые всеми его потомками: все классы элементов управления являются наследниками класса **Control**. Чаще всего, это прямые наследники, но иногда они имеют и непосредственного родителя, которым может быть абстрактный класс – это верно для кнопок, списков, текстовых элементов управления. Класс **Form** также является одним из потомков класса **Control**, т.е. форма – это тоже элемент управления, но со специальными свойствами. Будучи наследником классов **ScrollableControl** и **ContainerControl**, форма допускает прокрутку и размещение элементов управления.

В табл. 7 описаны основные свойства класса **Control**.

Таблица 7. Основные свойства класса **Control**

Свойство	Описание
BackColor	Возвращает или задаёт цвет фона для элемента управления
BackgroundImage	Возвращает или задаёт фоновое изображение, отображаемое на элементе управления
BackgroundImage-Layout	Возвращает или задаёт макет фонового изображения в соответствии с его определением в перечислении **ImageLayout** (**None**, **Title**, **Center**, **Stretch**, **Zoom**)
Bottom	Получает расстояние (в пикселях) между нижней границей элемента управления и верхней границей клиентской области контейнера
Bounds	Возвращает или задаёт размер и местоположение (в пикселях) элемента управления относительно его родительского элемента управления
CanFocus	Получает значение логического типа, показывающее, может ли элемент управления получать фокус
Focused	Получает значение, показывающее, имеется ли в элементе управления фокус ввода
Enabled	Возвращает или задаёт значение логического типа, показывающее, сможет ли элемент управления отвечать на действия пользователя
Font	Возвращает или задаёт шрифт текста, отображаемого элементом управления
ForeColor	Возвращает или задаёт основной цвет элемента управления
Height	Возвращает или задаёт высоту элемента управления (в пикселях)
Left	Возвращает или задаёт расстояние (в пикселях) между левой границей

	элемента управления и левой границей клиентской области его контейнера
Location	Возвращает или задаёт координаты левого верхнего угла элемента управления относительно левого верхнего угла контейнера (в полях **X** и **Y**)
Name	Возвращает или задаёт имя элемента управления
Size	Возвращает или задаёт высоту и ширину элемента управления (поля **Width** и **Height**)
TabIndex	Возвращает или задаёт последовательность перехода элемента управления внутри контейнера
TabStop	Возвращает или задаёт значение, показывающее, может ли пользователь перевести фокус в данный элемент управления клавишей **Tab**
Text	Возвращает или задаёт текст, сопоставленный с этим элементом управления (отображающийся в нём)
Top	Возвращает или задаёт расстояние (в точках) между верхней границей элемента управления и верхней границей клиентской области его контейнера
Visible	Возвращает или задаёт значение, указывающее отображаются ли элемент управления и все его родительские элементы управления
Width	Возвращает или задаёт ширину элемента управления

В числе событий класса **Control** можно отметить событие **Click**, которое наступает в том случае, когда происходит щелчок мышью на элементе управления (например, кнопке). Это событие наступает также и в том случае, если пользователь нажимает клавишу **Enter**.

4.3 Основные элементы управления

4.3.1 Элемент управления **TextBox** (текстовое поле)

Элемент управления **TextBox** позволяет пользователю вводить текст в приложение. Обычно элемент управления **TextBox** используется для отображения или ввода одной строки текста. Свойства **Multiline** и **ScrollBars** позволяют отображать или вводить несколько строк.

Свойство логического типа **ReadOnly** устанавливает, является ли текст в текстовом поле доступным только для чтения. Свойство **PasswordChar** позволяет маскировать знаки, вводимые в однострочную версию элемента управления. Чтобы ограничить объём текста, вводимого в **TextBox**, можно задать предельное число вводимых знаков как значение свойства **MaxLength**.

4.3.2 Элемент управления **Button** (кнопка)

Элементу управления «кнопка» соответствует класс **Button**. Работать с ним несложно. Обработки события **Click** оказывается достаточным для большинства приложений.

2.3.3 Элемент управления **Label** (надпись)

Элемент управления **Label** представляет собой надпись или короткую рекомендацию, объясняющую что-либо пользователю. Обычно для элемента **Label** не требуется обработка событий. Однако он обладает большим количеством свойств, в основном являющихся производными от класса **Control**, но есть и несколько новых (табл. 8).

Таблица 8. Основные свойства класса **Label**

Свойство	Описание
BorderStyle	Возвращает или задаёт стиль границы для элемента управления (**None**, **FixedSingle**, **Fixed3D**). По умолчанию установлено значение **BorderStyle.None**
FlatStyle	Возвращает или задаёт плоский внешний вид для элемента управления. Если свойство имеет значение **Flat**, элемент управления выглядит плоским; **Popup** – элемент управления выглядит плоским, но при наведении на него указателя мыши становится объемным; **Standard** – элемент управления выглядит объёмным (значение по умолчанию); **System** – внешний вид элемента управления определяется операционной системой
Image	Рисунок, выводимый на тексте
TextAlign	Выравнивание текста в элементе управления

4.3.4 Элемент управления **Panel** (панель)

Объект **Panel** – это элемент управления, несущий на себе другие элементы управления. Он не имеет заголовка. Если для свойства **Enabled** элемента управления **Panel** устанавливается значение **false**, элементы управления, содержащиеся в **Panel**, также будут отключены.

Элемент управления **Panel** по умолчанию отображается без границ. Свойство **BorderStyle** позволяет отобразить стандартную или трёхмерную границу, чтобы выделить область панели среди других областей на форме. Поскольку элемент управления **Panel** выводится из класса **ScrollableControl**, с помощью свойства **AutoScroll** можно включить на панели полосы прокрутки.

Задание

Часть I

1) Изучить интегрированную среду **Visual Studio** и освоить методы эффективной разработки и отладки программ в ней (раздел 1).

2) Изучить основные элементы управления **Windows Forms** (подраздел 4.3), основные типы данных языка **C#** и работу с ними (подразделы 3.1, 3.2, 3.4, 3.5)

3) Создать приложение **Windows Forms** для платформы **.NET Framework**. Поместить на форму две панели (элемент управления **Panel**), на 1-й разместить три поля **TextBox** (для ввода двух операндов и операции), на 2-й – элемент **Button**, запускающий вычисления, и элемент **Label** для вывода результата.

4) Установить для панелей различные значения свойства **BorderStyle**.

5) Сконструировать калькулятор для трёх арифметических операций. Для операндов и результата использовать следующие типы данных:

Вариант 1: **short**, **double**.

Вариант 2: **long**, **double**.

Вариант 3: **int**, **double**.

Вариант 4: **sbyte**, **double**.

Вариант 5: **ushort**, **double**.

Вариант 6: **ulong**, **double**.

Вариант 7: **uint**, **double**.

Вариант 8: **byte**, **double**.

Вариант 9: **short**, **float**.

Вариант 10: **long**, **float**.

Вариант 11: **int**, **float**.

Вариант 12: **sbyte**, **float**.

Вариант 13: **ushort**, **float**.

Вариант 14: **ulong**, **float**.

Вариант 15: **uint**, **float**.

Вариант 16: **byte**, **float**.

Вариант 17: **short**, **decimal**.

Вариант 18: **long**, **decimal**.

Вариант 19: **int**, **decimal**.

Вариант 20: **sbyte**, **decimal**.

Вариант 21: **ushort**, **decimal**.

Вариант 22: **ulong**, **decimal**.

Вариант 23: **uint**, **decimal**.

Вариант 24: **byte**, **decimal**.

Применять при реализации калькулятора условный оператор

Вариант а: **if-else**.

Вариант б: **switch-case**.

Вариант в: **?:**.

В одном из объявлений переменных использовать неявное типизирование.

6) Выполнить, при необходимости, предложенные действия в интегрированной среде, реализовать дополнительную функциональность.

Часть II

1) Изучить (подраздел 4.2) и при загрузке формы установить следующие свойства класса **Control** или **Label** (для самой формы или размещённых на ней элементов):

Вариант 1: **Bounds**, **TabStop**.

Вариант 2: **Bottom**, **Font**.

Вариант 3: **Left**, **BackColor**.

Вариант 4: **Location**, **ForeColor**.

Вариант 5: **Size**, **TabIndex**.

Вариант 6: **Width**, **TextAlign**.

Вариант 7: **BackgroundImageLayout**, **FlatStyle**.

Вариант 8: **BorderStyle**, **Top**.

Вариант 9: **BackgroundImage**, **Height**.

2) Изучить тип (класс) **string** языка C#, освоить его методы (пункт 3.3.1).

3) Перед вычислениями применить к операндам и операции строковые методы

Вариант I: **Trim**().

Вариант II: **TrimStart**()/**TrimEnd**().

4) Реализовать дополнительную операцию калькулятора (вводить для неё собственное условное обозначение – буквенно-цифровую комбинацию), используя указанный строковый метод:

Вариант А: **Insert**().

Вариант Б: **Replace**().

Вариант В: **Remove**().

Вариант Г: **Substring**().

Вариант Д: **Split**().

Вариант Е: **PadLeft**().

Вариант Ж: **ToCharArray**().

Вариант И: **PadRight**().

Первый операнд при этом должен иметь строковый тип.

5) Изучить особенности работы с несколькими формами (подраздел 4.1) и добавить в приложение 2-ую форму. Разместить на ней элемент управления **Label**.

6) Создать экземпляр 2-й формы и продублировать на нём результат вычислений.

7) Реализовать, при необходимости, дополнительную функциональность, предложенную преподавателем.

Примечание. В этой и последующих работах следует использовать осмысленные имена элементов управления и переменных, указывающие как на назначение, так и на класс объекта (например, в названии labelRes «label» указывает на класс **Label**, «Res» – на назначение, вывод результата вычислений).