
С. М. СТАНКЕВИЧ

Технологии программирования

Лабораторная работа № 1

Программирование на языке C#
алгоритмов следования и ветвления



С. М. СТАНКЕВИЧ

Технологии программирования

Лабораторная работа № 1

Программирование на языке C#
алгоритмов следования и ветвления

*Учебно-методическое пособие для студентов,
обучающихся по специальности «Прикладная информатика
(программное обеспечение компьютерных систем)»*

Витебск, 2020

Содержание

Введение	2
1. Интегрированная среда разработки приложений	
<i>Microsoft Visual Studio</i>	3
1.1. Общие сведения о <i>Microsoft Visual Studio</i>	3
1.2. Структура проекта	3
1.3. Работа с проектом	6
1.4. Настройка формы	9
1.5. Размещение элементов управления на форме	9
1.6. Написание кода обработки события	10
1.7. Запуск приложения	12
1.8. Динамическое изменение свойств	13
2. Программирование алгоритмов следования	15
2.1. Описание данных	15
2.2. Ввод и вывод данных в программе	17
2.3. Консольный ввод и вывод	19
2.4. Операции	20
2.5. Пример приложения для вычисления сложной функции	23
2.6. Выполнение индивидуального задания	25
3. Программирование алгоритмов ветвления	29
3.1. Условные операторы	29
3.2. Элемент управления RadioButton	32
3.3. Пример приложения для вычисления сложной функции	33
3.4. Выполнение индивидуального задания	34
Список использованных источников	36

Введение

Целью лабораторной работы является создание проектов консольного и оконного приложений с помощью IDE Microsoft Visual Studio.

Задачи лабораторной работы:

- изучить приёмы работы и методы настройки интегрированной среды разработки приложений *Microsoft Visual Studio*;
- научиться размещать элементы управления на форме и настраивать их внешний вид;
- научиться программировать элементы управления для организации работы приложения;
- научиться составлять каркас простейших консольных и оконных приложений в среде *Microsoft Visual Studio*;
- написать и отладить программу:
 - 1) линейного алгоритма;
 - 2) разветвляющегося алгоритма.

1. Интегрированная среда разработки приложений

Microsoft Visual Studio

1.1. Общие сведения о *Microsoft Visual Studio*

Microsoft Visual Studio— линейка продуктов компании *Microsoft*, включающих интегрированную среду разработки программного обеспечения и ряд других инструментальных средств. Данные продукты позволяют разрабатывать как консольные приложения, так и приложения с графическим интерфейсом, в том числе с поддержкой технологии *Windows Forms*, а также веб-сайты, веб-приложения, веб-службы как в родном, так и в управляемом кодах для всех платформ, поддерживаемых *Windows*, *Windows Mobile*, *Windows CE*, *.NET Framework*, *Xbox* и др.

Среда *MS Visual Studio* визуально реализуется в виде одного окна с несколькими панелями инструментов. Количество, расположение, размер и вид панелей может меняться программистом или самой средой разработки в зависимости от текущего режима работы среды или пожеланий программиста, что значительно повышает производительность работы.

Среда *MS Visual Studio* включает в себя редактор исходного кода с поддержкой технологии *IntelliSense* и возможностью простейшего рефакторинга кода. Встроенный отладчик может работать как отладчик уровня исходного кода, так и отладчик машинного уровня. Остальные встраиваемые инструменты включают в себя редактор форм для упрощения создания графического интерфейса приложения, веб-редактор, дизайнер классов и дизайнер схемы базы данных. *IDE MS Visual Studio* позволяет создавать и подключать сторонние дополнения (плагины) для расширения функциональности практически на каждом уровне, включая добавление поддержки систем контроля версий исходного кода (как, например, *Subversion* и *Visual SourceSafe*), добавление новых наборов инструментов (например, для редактирования и визуального проектирования кода на предметно-ориентированных языках программирования) или инструментов для прочих аспектов процесса разработки программного обеспечения.

1.2. Структура проекта

Перед началом программирования необходимо создать *проект*. Проект содержит все исходные материалы для приложения, такие как файлы исходного кода, ресурсов, значки, ссылки на внешние файлы, на которые опирается программа, и данные конфигурации, такие как параметры компилятора.

Кроме понятия проект часто используется более глобальное понятие — *решение (solution)*. Решение содержит один или несколько проектов, один из которых может быть указан в качестве стартового. С него будет начинаться выполнение решения. Таким об-

разом, при создании простейшей C#-программы в *MS Visual Studio* создается папка решения, в которой для каждого проекта создаётся подпапка проекта, а уже в ней будут находиться другие подпапки с результатами компиляции приложения.

Проект — это основная единица, с которой работает программист. При создании проекта можно выбрать его тип, а *MS Visual Studio* создаст каркас проекта в соответствии с выбранным типом.

Проект приложения с графическим интерфейсом в *MS Visual Studio* состоит из *файла проекта* (файл с расширением `.csproj`), одного или нескольких *файлов исходного текста* (с расширением `.cs`), *файлов с описанием форм* (с расширением `.Designer.cs`), *файлов ресурсов* (с расширением `.resx`), а также ряда служебных файлов (рис. 1).

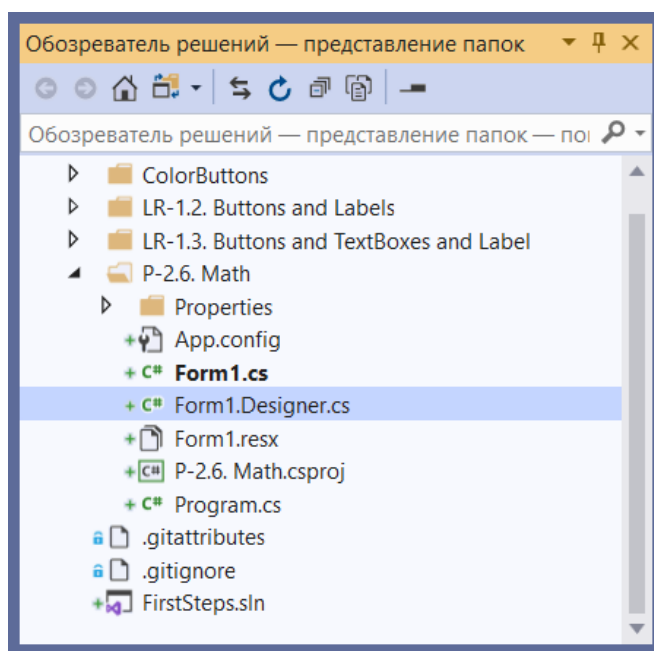


Рис. 1 — Структура проекта в окне *Обозревателя решений*

В файле проекта находится информация о модулях, составляющих данный проект, входящих в него ресурсах, а также параметров построения программы. Файл проекта автоматически создаётся и изменяется средой *MS Visual Studio* и не предназначен для ручного редактирования.

Файл исходного текста — программный модуль, предназначенный для размещения текстов программ. В этом файле программист размещает текст программы, написанной на языке C#. Модуль имеет следующую структуру:

```
// Раздел подключенных пространств имен
using System;

// Пространство имен текущего проекта
namespace MyFirstApp
{
    // Класс окна
    public partial class Form1 : Form
```

```

{
    // Методы окна
    public Form1()
    {
        InitializeComponent();
    }
}
}

```

В разделе подключения пространств имён (каждая строка которого располагается в начале файла и начинается ключевым словом **using**) описываются используемые *пространства имён*. Каждое пространство имён включает в себя классы, выполняющие определённую работу. Например, классы для работы с сетью располагаются в пространстве **System.Net**, а для работы с файлами — в **System.IO**. Большая часть пространств, которые используются в обычных проектах, уже подключена при создании нового проекта, но при необходимости можно дописать дополнительные пространства имён.

Чтобы не происходило конфликтов имён классов и переменных, классы проекта также помещаются в отдельное пространство имён. Оно определяется ключевым словом **namespace**, после которого следует имя пространства (обычно оно совпадает с именем проекта).

Внутри пространства имен помещаются *классы* — в новом проекте это класс окна, который содержит все методы для управления поведением окна. В определении класса присутствует ключевое слово **partial**, которое указывает, что в исходном тексте представлена только часть класса, с которой производится непосредственно работа. Служебные методы для обслуживания окна скрыты в другом модуле (при желании их тоже можно посмотреть, но редактировать вручную их не рекомендуется).

Внутри класса располагаются переменные, методы и другие элементы программы. Фактически, основная часть программы размещается внутри класса при создании обработчиков событий.

При компиляции программы *MS Visual Studio* создает исполняемые **.exe**-файлы в каталоге **bin**.

Кроме приложений с графическим интерфейсом (или иначе приложений *Windows Forms*) язык **C#** позволяет создавать проекты другого типа — *консольные приложения*.

По своим «внешним» проявлениям консольные приложения напоминают приложения *DOS*, запущенные в *Windows*. Тем не менее, это настоящие Win32-приложения, которые под *DOS* работать не будут. Для консольных приложений доступен *Win32 API*, а кроме того, они могут использовать *консоль* — окно, предоставляемое системой, которое работает в текстовом режиме и в которое можно вводить данные с клавиатуры. Особенность консольных приложений в том, что они работают не в графическом, а в текстовом режиме.

Программный модуль консольного приложения имеет следующую структуру:

```

using System;


```

```
namespace ConsoleApp
{
    class Program
    {
        static void Main(string[] args) {
            Console.WriteLine("Привет, мир!");
        }
    }
}
```

1.3. Работа с проектом

Так как проект в *MS Visual Studio* состоит из многих файлов, по умолчанию при создании нового проекта *MS Visual Studio* сохраняет его в отдельной папке.

Сразу после создания проекта рекомендуется сохранить его в подготовленной папке командой **Файл ► Сохранить всё**. При внесении значительных изменений в проект следует ещё раз сохранить проект той же командой, а перед запуском программы на выполнение среда обычно сама сохраняет проект на случай какого-либо сбоя. Для открытия существующего проекта используется команда **Файл ► Открыть проект**, либо можно найти в папке файл проекта с расширением `.sln` и запустить его стандартным для операционной системы способом.

При запуске IDE *MS Visual Studio* появляется начальная страница со списком последних проектов, а также командами **Создать проект** и **Открыть проект**. Кроме этого, создать новый проект можно, нажав кнопку  **Создать проект** на панели *Стандартная* или выбрав команду **Файл ► Создать проект** или нажав сочетание клавиш **Ctrl**+**Shift**+**N**. В этом случае на экране появится диалог для создания нового проекта (рис. 2).

В правой части из выпадающих списков нужно выбрать требуемый язык программирования (**C#** в данном случае), программную платформу (*Windows* в данном случае) и тип проекта (*Рабочий стол* в данном случае). Далее в списке проектов конкретизируется платформа проекта (в данном случае — *Приложение Windows Forms (.NET Framework)*).

Далее в лабораторных работах будут использоваться два типа проектов:

- *Приложение Windows Forms (.NET Framework)* — данный тип проекта позволяет создать полноценное приложение с окнами и элементами управления (кнопками, полями ввода и пр.). Такой вид приложения наиболее привычен большинству пользователей.

- *Консольное приложение (.NET Framework)* — в этом типе проекта окно представляет собой текстовую консоль, в которую приложение может выводить тексты или ожидать ввода информации пользователя. Консольные приложения часто используются для вычислительных задач, для которых не требуется сложный или красивый пользовательский интерфейс.

После нажатия кнопки **Далее** в открывшемся окне нужно более детально настроить параметры проекта (рис. 3): его название, каталог для сохранения и другие.

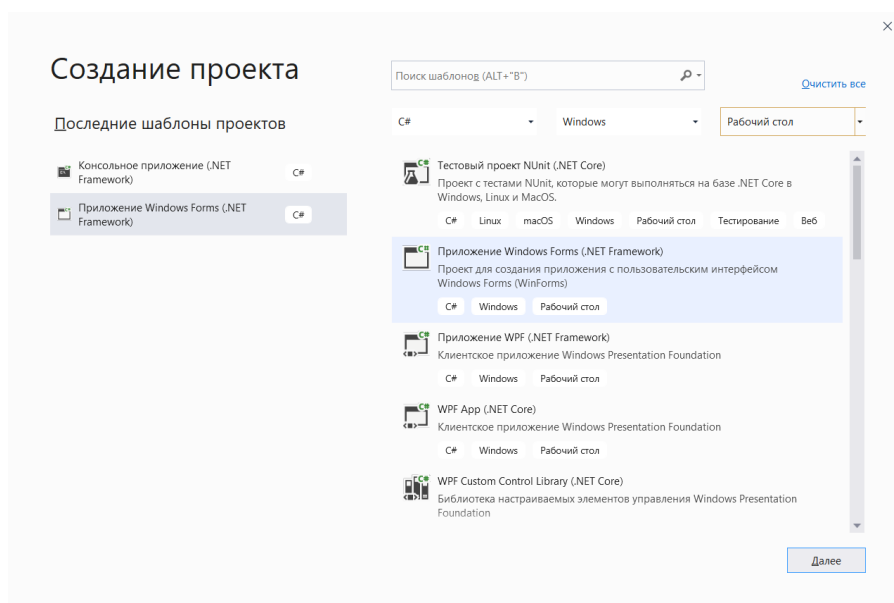


Рис. 2 — Диалог создания нового проекта

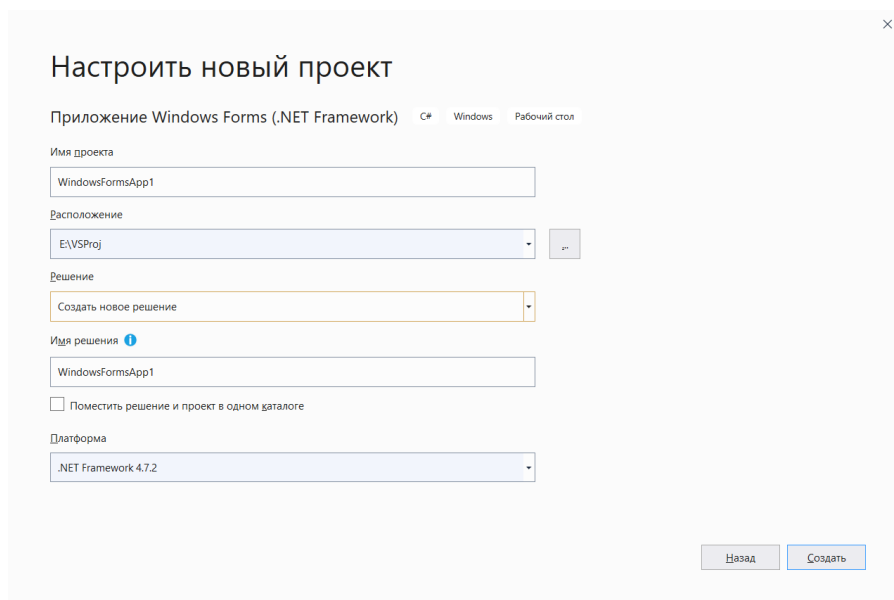


Рис. 3 — Окно с параметрами настройки проекта

После нажатия кнопки **Создать** через несколько секунд *MS Visual Studio* создаст проект и откроет его в режиме конструктора главного окна приложения (рис. 4).

В главном окне *MS Visual Studio* присутствует несколько основных элементов (см. рис. 4). Прежде всего, это **окно редактирования** (1) — здесь на вкладках отображаются будущее окно проектируемого приложения (вкладка конструктора форм), на котором будут размещаться элементы управления, и код приложения. При выполнении программы размещённые на форме элементы управления будут иметь тот же вид, что и на этапе проектирования.

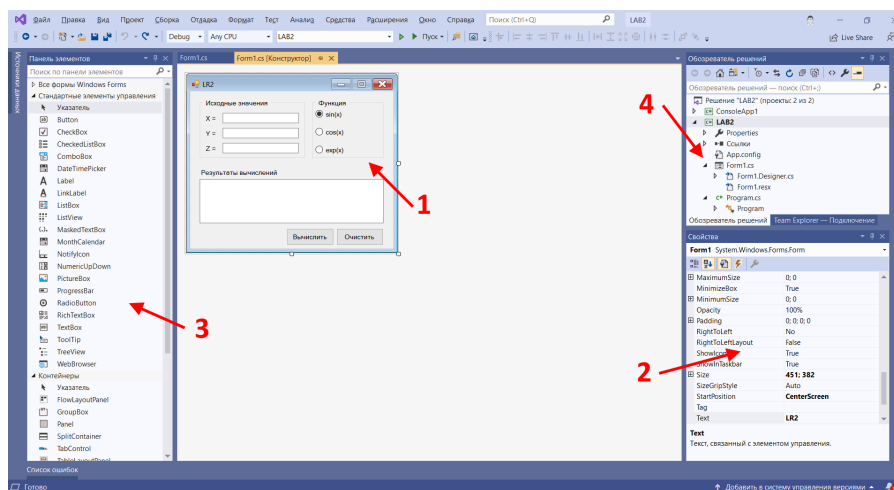


Рис. 4 — Главное окно IDE MS Visual Studio с открытым конструктором окна приложения

Второй по важности объект — это окно **Свойства** (2), в котором перечислены все свойства выделенного элемента управления или окна. С помощью кнопки **Свойства** можно просматривать свойства элемента управления. Кнопка **События** переключает окно в режим просмотра списка событий, которые может обрабатывать данный элемент управления. Для ускорения поиска нужных свойств в списке его можно отобразить отсортированным по алфавиту (с помощью кнопки **В алфавитном порядке**) или в соответствии с категориями свойств (с помощью кнопки **По категориям**). Если этого окна на экране нет, его можно активировать в меню командой **Вид ► Окно свойств** или комбинацией клавиш **Ctrl+W, P**.

Сами элементы управления можно перетаскивать или выбирать двойным щелчком мыши из **Панели элементов** (3). Все элементы управления разбиты на логические группы, что облегчает поиск нужных элементов. Если панели нет на экране, её нужно активировать командой **Вид ► Панель элементов** (**Ctrl+W, X**).

В окне **Обозревателя решений** (4) отображается список всех файлов, входящих в проект, включая добавленные изображения и служебные файлы. С помощью кнопки **Переключить представления** в окне можно отображать либо структуру проекта, либо иерархию папок файловой системы, в которых размещается текущий проект. Активируется окно обозревателя командой **Вид ► Обозреватель решений** (**Ctrl+W, S**).

Указанные панели могут уже присутствовать на экране, но быть скрытыми за другими панелями или свёрнуты к боковой стороне окна. В этом случае достаточно щёлкнуть на соответствующем ярлычке, чтобы вывести панель на передний план.

Окно редактирования предназначено для просмотра, написания и редактирования кода. Переключаться между конструктором формы и кодом можно с помощью команд **Вид ► Код** (**F7**) и **Вид ► Конструктор** (**Shift+F7**). При первоначальной загрузке в окне редактирования находится текст, содержащий минимальный набор операторов для нормального функционирования пустой формы в качестве Windows-окна. При помеще-

нии элемента управления в окно формы, текст программы автоматически дополняется описанием необходимых для его работы библиотек стандартных программ (перечисляемых в разделе `using`) и переменных для доступа к элементу управления (в скрытой части класса формы).

1.4. Настройка формы

Настройка формы обычно начинается с установки размеров формы. Эта операция выполняется путём перемещения с помощью мыши специальных белых маркеров, расположенных на правой и нижней сторонах формы.

Для задания любых свойств формы и элементов управления на форме используется окно **Свойства**. Новая форма имеет одинаковые имя (свойство `Name`) и заголовок (свойство `Text`) — `Form1`. Для изменения заголовка нужно щёлкнуть кнопкой мыши на форме, в окне свойств найти в списке свойство `Text` и после щелчка мыши в выделенном поле набрать требуемое название.

Также часто используется свойство `StartPosition`, установка которого в значение `CenterScreen` обеспечивает при запуске приложения появление формы в центре экрана компьютера.






1.5. Размещение элементов управления на форме

Все возможные элементы управления перечислены в **Панели элементов** (рис. 5), где сгруппированы по типу. Каждую группу элементов управления можно свернуть, если она в настоящий момент не нужна. В лабораторных работах в основном используются элементы управления из группы **Стандартные элементы управления**.

Для добавления элемента управления на форму нужно щёлкнуть на нём, а затем — на форме, или просто перетащить элемент из **Панели элементов** в нужное место формы. Затем его можно перемещать по форме с помощью левой кнопкой мышки (иногда это можно сделать лишь за появляющийся при нажатии на элемент квадрат со стрелками). Если на сторонах элемента управления при его выделении появляются белые маркеры, это означает, что с помощью мыши можно изменить размер элемента.

Доступ к параметрам в окне **Свойства** появляется после выделения соответствующего элемента (или группы элементов) на форме.

Наиболее часто на формах используются следующие элементы управления:

-  `TextBox` — для ввода из формы в программу или вывода на форму информации;
-  `Label` — для размещения пояснительных надписей;
-  `RadioButton` — для выбора альтернативных вариантов использования;
-  `CheckBox` — для выбора нескольких вариантов использования;
-  `Button` — для запуска различных действий на выполнение.

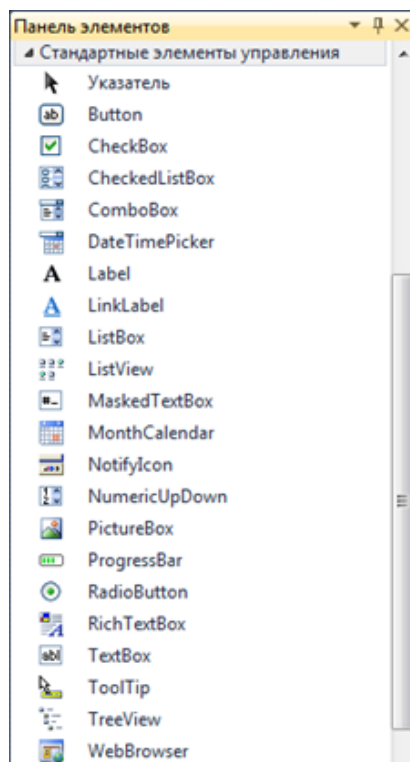



Рис. 5 — Панель элементов


После размещения элемента управления на форме в тексте программы можно использовать связанную с ним переменную (например, `textBox1` для элемента  `TextBox`) и все доступные свойства. Переменную при необходимости можно переименовать.

1.6. Написание кода обработки события


Программирование поведения приложения *Windows Forms* на языке *C#* подчиняется событийно-ориентированной парадигме программирования и заключается в описании алгоритмов, которые необходимо выполнять при возникновении определённых событий, связанных с приложением (например, загрузка данных в приложение, печать формы, аварийное завершение приложения и т. д.).

С каждым элементом управления на форме и с самой формой также связаны некоторые события, которые возникают во время работы приложения (например, нажатие кнопки, изменение размеров окна, ввод текста в текстовое поле и т. д.). Эти события обрабатываются в приложении с помощью *обработчиков событий* — специальных *методов* соответствующих элементов управления.

Для создания обработчика события существует два способа.

Первый способ — создать обработчик для события по умолчанию (обычно это самое часто используемое событие данного элемента управления). Например, для элемента управления  `Button` таким событием является нажатие на кнопку (щелчок левой кноп-

кой мыши в области кнопки). Соответствующий алгоритм обработки будет запрограммирован в теле метода `<Button>_Click()`, где `<Button>` — имя элемента управления.

Написание кода обработки события нажатия кнопки. Поместите на форму элемент управления  `Button`. С помощью окна **Свойства** измените надпись на кнопке (свойство `Text`) на слово «Привет» или другое по вашему желанию. Отрегулируйте положение и размер кнопки. После этого два раза щёлкните мышью на кнопке, в окне редактирования появится заготовка метода-обработчика события нажатия кнопки:


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

}
```

Далее можно добавлять необходимый код между скобками `{ · }`. Например, наберите:


```
private void button1_Click(object sender, EventArgs e)
{
    MessageBox.Show("Привет, " + textBox1.Text + "!");
}
```

Написание кода обработки события загрузки формы. Другой способ создания обработчика события заключается в выборе из списка возможных методов нужного события для выделенного элемента на форме.

Выделите форму. В окне **Свойства** на вкладке  **События** найдите в списке событие `Load`. Двойным щелчком откройте заготовку кода метода-обработчика события, между скобками `{ · }` которого вставьте текст программы, как показано ниже:

```
private void Form1_Load(object sender, EventArgs e)
{
    BackColor = Color.AntiqueWhite;
}
```


Обратите внимание!

*Если какой-то обработчик был добавлен по ошибке или больше не нужен, то для его удаления нельзя просто удалить программный код метода обработки события. Сначала нужно удалить строку с именем события в списке в окне **Свойства** на вкладке  **События**. В противном случае программа может перестать компилироваться.*

Каждый элемент управления имеет свой набор обработчиков событий, однако некоторые из них присущи большинству элементов управления. Наиболее часто используемые события представлены в следующей таблице:

Событие	Описание события
Activated	Форма получает это событие при активации.
Load	Возникает при загрузке формы. В обработчике данного события следует задавать действия, которые должны происходить в момент создания формы, например установка начальных значений.
KeyPress	Возникает при нажатии кнопки на клавиатуре. Параметр <code>e.KeyChar</code> имеет тип <code>char</code> и содержит код нажатой клавиши (клавиша <code>Enter</code> клавиатуры имеет код 13, клавиша <code>Esc</code> — код 27 и т. д.). Обычно это событие используется в том случае, когда необходима реакция на нажатие одной из клавиш.
KeyDown	Возникает при нажатии клавиши на клавиатуре. Обработчик этого события получает информацию о нажатой клавише и состоянии клавиш <code>Shift</code> , <code>Alt</code> и <code>Ctrl</code> , а также о нажатой кнопке мыши. Информация о клавише передается параметром <code>e.KeyCode</code> , который представляет собой перечисление <code>Keys</code> с кодами всех клавиш. Информацию о клавишах-модификаторах можно узнать из параметра <code>e.Modifiers</code> .
KeyUp	Является парным событием для <code>KeyDown</code> и возникает при отпускании ранее нажатой клавиши.
Click	Возникает при нажатии кнопки мыши в области элемента управления.
DoubleClick	Возникает при двойном нажатии кнопки мыши в области элемента управления.

1.7. Запуск приложения

Запустить приложение можно командой меню **Отладка** ► **Начать отладку** или кнопкой  **Пуск** на панели инструментов. При этом происходит трансляция и, если нет ошибок, компоновка программы и создание единого загружаемого файла с расширением `.exe`. На экране появляется активное окно приложения.

Если в программе есть ошибки, то в окне **Список ошибок** появятся найденные ошибки, а программа в таком случае обычно не запускается. Иногда, однако, может быть запущена предыдущая версия программы, в которой ещё нет последних изменений. Чтобы этого не происходило, нужно в настройках *MS Visual Studio* (команда меню **Средства** ► **Параметры**) установить опцию показа окна ошибок и запретить запуск при наличии ошибок (рис. 6–7).

Для завершения работы программы и возвращения в режим конструирования формы окно приложения нужно закрыть предусмотренными средствами (специальной командой, стандартной кнопкой в заголовке окна и др.).

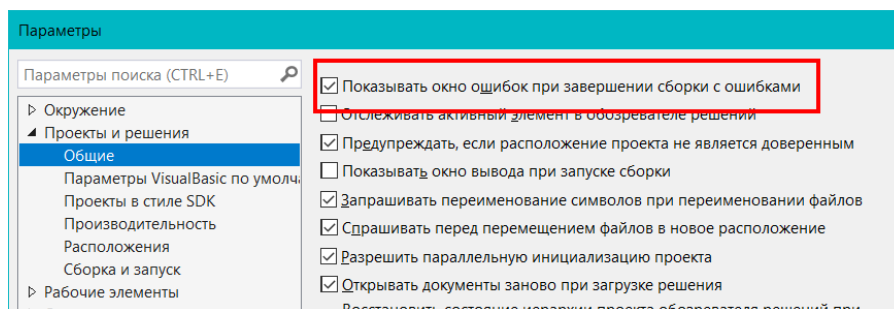


Рис. 6 — Опция показа сообщений об ошибках

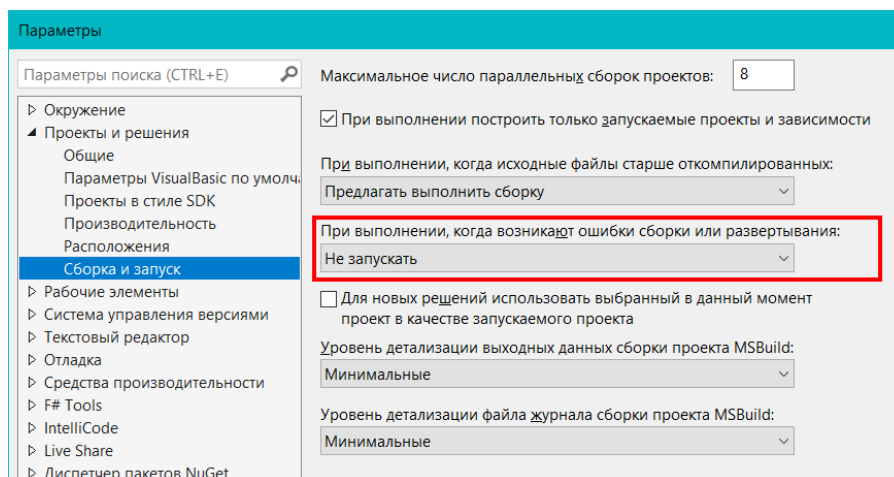


Рис. 7 — Опция отключения запуска предыдущей версии при ошибках построения

1.8. Динамическое изменение свойств

Свойства элементов управления окна могут быть изменены динамически во время выполнения программы. Например, можно изменить текст надписи или цвет формы. Изменение свойств происходит внутри обработчика события (например, обработчика события нажатия на кнопку). Для этого используют оператор присвоения вида:

```
<имя элемента>.<свойство> = <значение>;
```

Например:

```
label1.Text = "Привет";
```

Параметр <имя элемента> определяется на этапе проектирования формы, при размещении элемента управления на форме. Например, при размещении на форме ряда элементов `TextBox`, эти элементы получают имена `textBox1`, `textBox2`, `textBox3` и т. д. Эти имена могут быть заменены в окне **Свойства** в свойстве (**Name**) для текущего элемента. Допускается использование латинских или русских символов, знака подчёркивания и цифр (цифра не должна стоять в начале идентификатора). Список свойств для конкретного

элемента можно посмотреть в окне **Свойства**, а также в приложении к данным методическим указаниям.

Если требуется изменить свойства формы, то никакое имя элемента перед точкой вставлять не нужно, как и саму точку. Например, чтобы задать цвет формы, нужно просто написать:

```
BackColor = Color.Green;
```


2. Программирование алгоритмов следования

2.1. Описание данных

Типы данных имеют особенное значение в `C#`, поскольку это строго типизированный язык. Это означает, что все операции подвергаются строгому контролю со стороны компилятора на соответствие типов, причём недопустимые операции не компилируются. Такая строгая проверка типов позволяет предотвратить ошибки и повысить надёжность программ.

На рис. 8 показана иерархия типов языка `C#`.

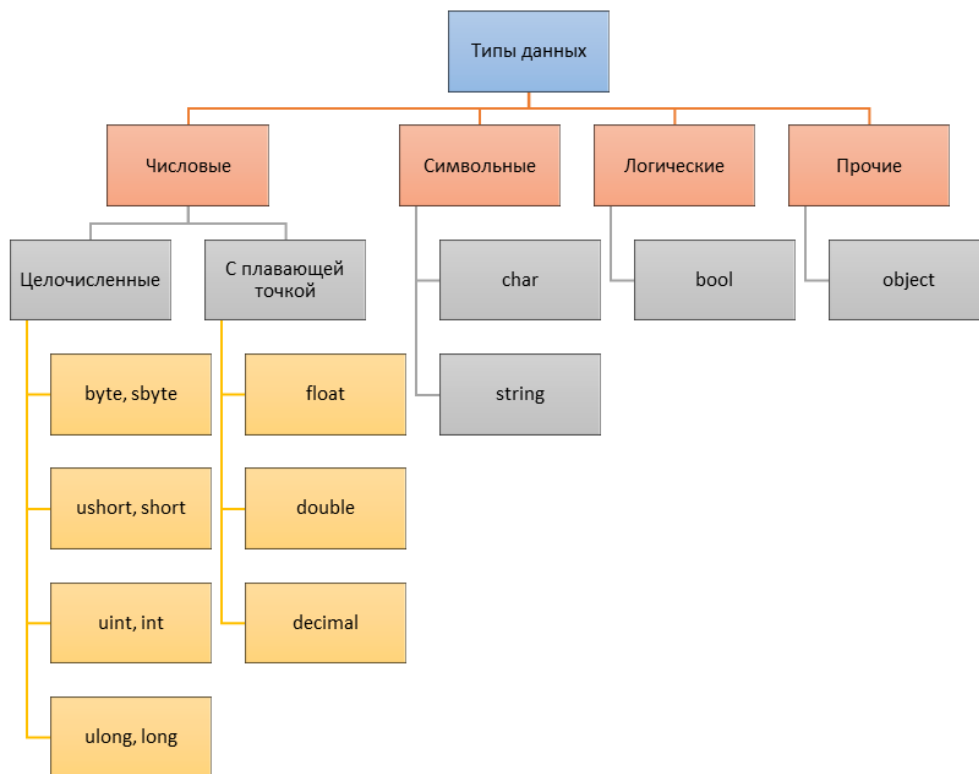


Рис. 8 — Иерархия типов данных языка `C#`

Для обеспечения контроля типов все переменные, выражения и значения должны принадлежать к определённому типу. Такого понятия, как бестиповая переменная, допустимая в ряде скриптовых языков, в `C#` вообще не существует. Более того, тип значения определяет те операции, которые разрешается выполнять над ним. Операция, разрешённая для одного типа данных, может оказаться недопустимой для другого.

Целочисленные типы. В C# определены девять целочисленных типов: `char`, `byte`, `sbyte`, `short`, `ushort`, `int`, `uint`, `long` и `ulong`. Тип `char` может хранить числа, но чаще используется для представления символов. Остальные восемь целочисленных типов предназначены для числовых расчётов. Некоторые целочисленные типы могут хранить как положительные, так и отрицательные значения (`sbyte`, `short`, `int` и `long`), другие же — только положительные (`char`, `byte`, `ushort`, `uint` и `ulong`).

Типы с плавающей точкой. В C# имеются три разновидности типов данных с плавающей точкой: `float`, `double` и `decimal`. Такие типы позволяют представлять числа с дробной частью. Первые два типа представляют числовые значения с одинарной и двойной точностью, вычисления над ними выполняются аппаратно и поэтому быстро. Тип `decimal` служит для представления чисел с плавающей точкой высокой точности без округления, характерного для типов `float` и `double`. Вычисления с использованием этого типа выполняются программно и поэтому более медленны.

Числовые литералы, входящие в выражения, по умолчанию интерпретируются как целочисленные. Поэтому следующее выражение будет иметь значение 0, ведь если 1 нацело разделить на 2, то получится как раз 0:

```
double x = 1 / 2;
```

Чтобы этого не происходило, в подобных случаях нужно явно указывать тип чисел с помощью символов-модификаторов: «f» для `float` и «d» для `double`. Приведённый выше пример правильно будет выглядеть так:

```
double x = 1d / 2d;
```

Часто в программе возникает необходимость записать число в экспоненциальной форме. Для этого после мантиссы числа записывается символ «e» и сразу после него — порядок. Например, число $2,5 \cdot 10^{-2}$ будет записано в программе следующим образом:

```
double x = 2.5e-2;
```

Символьные типы. В C# символы представлены 16-разрядной кодировкой Юникод (Unicode, UTF-16). В кодировке Юникод набор символов представлен настолько широко, что он охватывает символы практически из всех естественных языков, существующих в мире.

Основным типом при работе со строками является тип `string`, задающий строки переменной длины. Тип `string` представляет последовательность из нуля или более символов в кодировке Юникод. По сути, текст хранится в виде последовательной коллекции объектов `char`, доступной только для чтения.

Логический тип данных. Тип `bool` представляет два логических значения: «Истина» и «Ложь». Эти логические значения обозначаются в `C#` зарезервированными словами `true` и `false` соответственно. Следовательно, переменная или выражение типа `bool` будет принимать одно из этих логических значений.

Переменные и константы. *Переменная* — это ячейка памяти, которой присвоено некоторое имя и это имя используется для доступа к данным, расположенным в этой ячейке. Диапазон всех возможных значений для данной переменной определяется *типом данных*. Переменные объявляются непосредственно в тексте программы. Лучше всего сразу присвоить им начальное значение с помощью оператора присваивания «`=`»:

```
int a;           // Только объявление
int b = 7;       // Объявление и инициализация
```

Для того чтобы присвоить значение символьной переменной, достаточно заключить это значение (т. е. символ) в одинарные кавычки:

```
char ch;           // Только объявление
char symbol = Z;   // Объявление и инициализация
```

а строковой — в двойные кавычки:

```
string str;        // Только объявление
string str2 = "ABC"; // Объявление и инициализация
```

Частным случаем переменных являются константы. *Константы* — это переменные, значения которых не меняются в процессе выполнения программы. Объявление константы предваряется ключевым словом `const`:

```
const int c = 5;
```

2.2. Ввод и вывод данных в программе

Рассмотрим один из способов ввода данных через элементы, размещённые на форме. Для ввода данных чаще всего используют элемент управления `TextBox` через обращение к его свойству `Text`, которое хранит в себе строку введённых символов. Поэтому данные можно считать таким образом:

```
private void button1_Click(object sender, EventArgs e)
{
    string s = textBox1.Text;
}
```

Однако со строкой символов невозможно производить арифметические операции. Поэтому лучше всего при вводе числовых данных осуществлять преобразование строки в число требуемого типа. Для этого у объектов, представляющие типы, существует метод `Parse`. Поэтому предыдущий пример можно переделать следующим образом:

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

```
{
    int a = int.Parse(textBox1.Text);
    int b = a * a;
}
```

Ещё один способ — воспользоваться методом объекта `Convert`:

```
private void button1_Click(object sender, EventArgs e)
{
    int a = Convert.ToInt32(textBox1.Text);
    int b = a * a;
}
```

Перед выводом числовых данных требуется выполнять обратное преобразование — чисел в строки. Для этого у каждого объекта, представляющего числовой тип данных, существует метод `ToString()`, который возвращает строку с символьным представлением значения. Вывод данных можно осуществлять, например, в элементы `TextBox` или `Label`, используя их свойство `Text`:

```
private void button1_Click(object sender, EventArgs e)
{
    int a = Convert.ToInt(textBox1.Text);
    int b = a * a;
    label1.Text = b.ToString();
}
```

В языках программирования в дробных числах чаще всего используется точка, например: 15.7. Однако в `C#` методы преобразования строк в числа (вроде `double.Parse()` или `Convert.ToFloat()`) учитывают региональные настройки *Windows*, в которых в качестве десятичной точки используется символ запятой (например, 15,7).

Формат вывода строки зависит от локали, установленной в операционной системе, под управлением которой выполняется приложение. Эта локаль неявно вызывается при форматировании значения и выводе на экран. Для того, чтобы в числах отображалась точка, можно явно указать американскую локаль ("`en-US`") для форматирования. Код будет выглядеть следующим образом:

```
label1.Text = b.ToString("0.00",
    System.Globalization.CultureInfo.GetCultureInfo("en-US"));
```

Лучше изменить значение для всей сборки, а не указывать локаль явно при каждом вызове метода `ToString()`. Тогда строку изменения локали можно вписать, например, в метод загрузки формы:

```
using System.Globalization;
...
private void Form1_Load(object sender, EventArgs e)
{
    initForm();
    CultureInfo.CurrentCulture = CultureInfo.GetCultureInfo("en-US");
}
```

}

Обратите внимание!

Для работы функции `GetCultureInfo()` требуется подключение пространства имён `System.Globalization`.

2.3. Консольный ввод и вывод

Для ввода данных в программу с клавиатуры служит метод `ReadLine()` объекта `Console`, который возвращает данные в виде строки:

```
string str = Console.ReadLine();
```

Далее полученную строку при необходимости следует преобразовать с использованием методов объекта `Convert`. Наиболее часто используемыми являются методы преобразования в целое число (`Convert.ToInt32()`) и в вещественные числа (`Convert.ToFloat()`, `Convert.ToDouble()`, `Convert.ToDecimal()`).

```
string str = Console.ReadLine();  
double x = Convert.ToDouble(str);
```

Для вывода строки или числа на консоль надо передать их в метод `WriteLine()` объекта `Console` в качестве параметра:

```
double a = 24.5;  
Console.WriteLine("Значение:");  
Console.WriteLine(a);
```

Нередко возникает необходимость вывести на консоль в одной строке значения сразу нескольких переменных. Для этого существует несколько способов.

Можно использовать *конкатенацию* строк — слияние нескольких строк в одну с помощью операции "+", как показано в примере ниже:

```
double a = 24.5;  
double b = a * a;  
Console.WriteLine("Значение и его квадрат равны " + a.ToString()  
                  + " и " + b.ToString() + ".");
```

В другом способе в методе `WriteLine()` указывают несколько параметров. Первый параметр представляет выводимую строку. Все последующие параметры представляют значения, которые требуется поместить в эту строку при выводе. При этом важен порядок следования подобных параметров.

В результирующей строке (первом параметре) место каждого выводимого значения обозначается т. н. *местодержателем* (*placeholder*) — порядковым номером значения в списке параметров, заключенным в фигурные скобки. Нумерация параметров начинается с нуля, как показано в примере ниже:

```
Console.WriteLine("Значение и его квадрат равны {0} и {1}.", a, b);
```

Ещё один способ формирования строки для вывода, появившийся в последних версиях языка C#, — использование *интерполяции*. В этом случае местозаменитель представляет собой имя переменной, заключенное в фигурные скобки, а вся строка предваряется символом доллара, как показано в примере ниже:

```
Console.WriteLine($"Значение и его квадрат равны {a} и {b}.");
```

Для вывода данных можно использовать и метод `Write()`. Его отличие от метода `WriteLine()` заключается в том, что при выводе переход на следующую строку не осуществляется.

2.4. Операции

Большинство операций языка C# имеют тот же смысл, как и в других языках программирования. *Операции* представляют определённые действия над *операндами* — участниками операции. В качестве операнда может выступать переменная или литерал. Операции бывают *унарными* (выполняются над одним операндом), *бинарными* — над двумя операндами и *тернарными* — над тремя операндами.

Далее в таблицах представлены некоторые стандартные операции языка C#.

Таб. 1 — Арифметические операции

Операция	Пояснение
+	Сложение
-	Вычитание
*	Умножение
/	Деление
%	Остаток от деления
++	Инкремент (увеличение на 1)
--	Декремент (уменьшение на 1)

Пример использования операций из таб. 1.

```
int x = 10;
int z;

z = x + 12;           // 22
z = x - 6;            // 4
z = x * 5;            // 50
int z = x / 5;        // 2

double a = 10;
double b = 3;
double c = a / b;     // 3.33333333

double m = 10 / 4;    // 2 (целочисленное деление)
```

Таб. 2 — Логические операции

Операция	Пояснение
&, &&	Логическое умножение (И)
,	Логическое сложение (ИЛИ)
!	Логическое отрицание (НЕ)
^	Логическое исключающее ИЛИ

Таб. 3 — Поразрядные логические операции

Операция	Пояснение
&	Логическое умножение (И)
	Логическое сложение (ИЛИ)
~	Логическое отрицание (НЕ)
^	Логическое исключающее ИЛИ
<<	Поразрядный сдвиг влево
>>	Поразрядный сдвиг вправо

```
int k = 10 % 4;      // 2
k++;                // 3
k--;                // 2
```

Две пары операций | и || (а также & и &&) выполняют похожие действия, однако они не равнозначны (см. таб. 2). Различие заключается в следующем.

В выражении `c = a | b` будут вычисляться оба значения `a` и `b`.

В выражении `c = a || b` сначала будет вычисляться значение `a`, и если оно равно `true`, то вычисление значения `b` уже смысла не имеет, т.к. в любом случае значение `c` будет равно `true`. Значение `b` будет вычисляться только в том случае, если `a` равно `false`. Это — пример т.н. «*ленивых вычислений*».

Подобным образом работает и пара операций & и &&. В выражении `c = a && b` значение `b` будет вычисляться только в том случае, когда значение `a = true`.

Переменные `a`, `b` и `c` в приведенных выше операциях имеют тип `bool`.

Пример использования операций из таб. 2.

```
bool x1 = (5 > 6) | (4 < 6); // (5 > 6 = false, 4 < 6 = true) => true
bool x3 = (5 > 6) & (4 < 6); // (5 > 6 = false, 4 < 6 = true) => false

bool x2 = (5 < 6) || (4 > 6); // (5 < 6 = true) => true
bool x4 = (5 > 6) && (4 < 6); // (5 > 6 = false) => false
```

Поразрядные логические операции (таб. 3) выполняются над отдельными разрядами чисел.

Пример использования операций из таб. 3.

```
int x1 = 2;    // 010
int y1 = 5;    // 101
Console.WriteLine(x1 & y1);    // 0
```

Таб. 4 — Операции сравнения

Операция	Пояснение
==	Равно
>	Больше
<	Меньше
>=	Больше либо равно
<=	Меньше либо равно

```

Console.WriteLine(x1 | y1);    // 111

int x = 45;                    // 101101
int key = 102;                 // 1100110
int encrypt = x ^ key;        // 1001011
Console.WriteLine("Зашифрованное число: {encrypt}");

int x = 12;                    // 00001100
Console.WriteLine(~x + 1);     // 11110100    или -12

```

Пример использования операций из таб. 4.

```

int a = 10;
int b = 4;
bool c = a == b;    // false

bool c = a != b;    // true
bool d = a!=10;     // false

bool c = a < b;      // false

bool c = a > b;      // true
bool d = a > 25;     // false

bool c = a <= b;     // false
bool d = a <= 25;    // true

bool c = a >= b;     // true
bool d = a >= 25;    // false

```

Пример использования операций из таб. 5.

```

int a = 10;
a += 10;    // 20
a -= 4;     // 16
a *= 2;     // 32
a /= 8;     // 4
a <<= 4;    // 64
a >>= 2;    // 16

```

Операции присвоения являются правоассоциативными, то есть выполняются справа налево. Например:

```
int a = 8;
```


Таб. 5 — Сокращённые операции присваивания

Операция	Пояснение
<code>+=</code>	Присваивание после сложения
<code>-=</code>	Присваивание после вычитания
<code>*=</code>	Присваивание после умножения
<code>/=</code>	Присваивание после деления
<code>%=</code>	Присваивание после вычисления остатка
<code>&=</code>	Присваивание после поразрядной конъюнкции
<code> =</code>	Присваивание после поразрядной дизъюнкции
<code>^=</code>	Присваивание после поразрядного исключающего ИЛИ
<code><<=</code>	Присваивание после сдвига разрядов влево
<code>>>=</code>	Присваивание после сдвига разрядов вправо

```
int b = 6;
int c = a += b -= 5;  // 9
```

В данном случае выполнение выражения будет идти следующим образом:

1. `b -= 5` (`6-5=1`)
2. `a += (b-=5)` (`8+1 = 9`)
3. `c = (a += (b-=5))` (`c = 9`)

Для задания приоритетов операций используются круглые скобки (`·`).

2.5. Пример приложения для вычисления сложной функции

Задание: составить программу вычисления для заданных значений x , y , z арифметического выражения

$$u = \operatorname{tg}^2(x + y) - e^{y-z} \sqrt{\cos x^2 + \sin z^2}.$$

Вариант размещения элементов управления в окне приложения показан на рис. 9.

Для вывода результатов вычислений в приложении используется элемент управления `TextBox`. Значение `True` свойства `Multiline` этого элемента позволит создать многострочный текст. После установки свойства `ScrollBars` в значение `Both` в текстовом поле появится вертикальная, а при необходимости и горизонтальная полосы прокрутки.

Информация, которая отображается построчно в окне, находится в массиве строк `Lines`, каждая строка которого имеет тип `string`. Однако нельзя напрямую обратиться

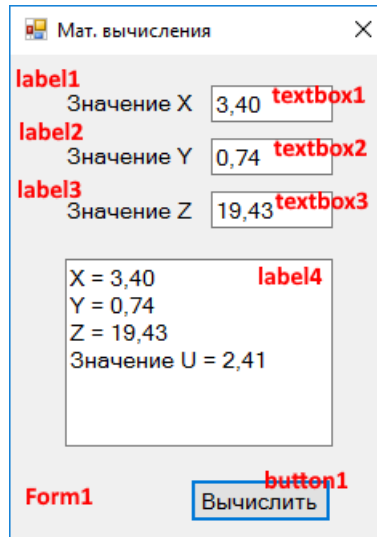


Рис. 9 — Вид приложения для вычисления сложной функции

к этому свойству для добавления новых строк, поскольку размер массивов в `C#` определяется в момент их инициализации. Для добавления нового элемента используется свойство `Text`, к текущему содержимому которого можно добавить новую строку:

```
textBox4.Text += Environment.NewLine + "Привет";
```

В этом примере к текущему содержимому текстового поля добавляется символ перевода курсора на новую строку (который может отличаться в разных операционных системах, и потому представлен константой `NewLine` класса `Environment`) и сама новая строка. Если добавляется числовое значение, то его предварительно нужно преобразовать в строковый вид методом `ToString()`.

Процесс вычислений запускается нажатием на кнопку `Вычислить`. В текстовом поле `textBox4` появляются исходные данные и результат вычислений. После изменений данных x , y , z в текстовых полях `textBox1`–`textBox3` для получения новых результатов снова нужно нажать кнопку `Вычислить`.

Далее приведен полный текст программы приложения для вычислений.

```
1 using System;
2 using System.Windows.Forms;
3
4 namespace MyFirstApp
5 {
6     public partial class Form1 : Form
7     {
8         public Form1()
9         {
10             InitializeComponent();
11         }
12
13         private void Form1_Load(object sender, EventArgs e)
14         {
```

```

15      // Начальное значение X
16      textBox1.Text = "3,4";
17
18      // Начальное значение Y
19      textBox2.Text = "0,74";
20
21      // Начальное значение Z
22      textBox3.Text = "19,43";
23  }
24
25  private void button1_Click(object sender, EventArgs e)
26  {
27      // Считывание значения X
28      double x = double.Parse(textBox1.Text);
29      // Вывод значения X в текстовое поле
30      textBox4.Text += Environment.NewLine + "X = " + x.ToString();
31
32      // Считывание значения Y
33      double y = double.Parse(textBox2.Text);
34      // Вывод значения Y в текстовое поле
35      textBox4.Text += Environment.NewLine + "Y = " + y.ToString();
36
37      // Считывание значения Z
38      double z = double.Parse(textBox3.Text);
39      // Вывод значения Z в текстовое поле
40      textBox4.Text += Environment.NewLine + "Z = " + z.ToString();
41
42      // Вычисление арифметического выражения
43      double a = Math.Tan(x + y) * Math.Tan(x + y);
44      double b = Math.Exp(y - z);
45      double c = Math.Sqrt(Math.Cos(x * x) + Math.Sin(z * z));
46      double u = a - b * c;
47
48      // Вывод результатов в текстовое поле
49      textBox4.Text += Environment.NewLine +
50          "Результат U = " + u.ToString();
51  }
52 }
53 }

```

2.6. Выполнение индивидуального задания

Ниже приведены варианты задания для самостоятельного выполнения. По указанию преподавателя выберите свой вариант. Уточните условие задания, количество, наименование, типы исходных данных. В соответствии с этим установите необходимое количество текстовых полей `TextBox`, тексты заголовков на форме, размеры шрифтов, а также типы переменных и функции преобразования при вводе и выводе результатов. Для проверки правильности программы после задания приведен контрольный пример: тестовые значения переменных, используемых в выражении, и результат, который при этом получается.

Вариант 1. $t = \frac{2 \cos \left(x - \frac{\pi}{6} \right)}{0,5 + \sin^2 y} \left(1 + \frac{z^2}{3 - \frac{z^2}{5}} \right).$

При $x = 14,26$, $y = -1,22$, $z = 3,5 \cdot 10^{-2}$ значение $t = 0,564849$.

Вариант 2. $u = \frac{\sqrt[3]{8 + |x - y|^2 + 1}}{x^2 + y^2 + 2} - e^{|x-y|}(\operatorname{tg}^2 z + 1)^x.$

При $x = -4,5$, $y = 0,75 \cdot 10^{-4}$, $z = 0,845$ значение $u = -55,6848$.

Вариант 3. $v = \frac{1 + \sin^2(x + y)}{\left| x - \frac{2y}{1 + x^2 y^2} \right|} x^{|y|} + \cos^2 \left(\operatorname{arctg} \frac{1}{z} \right).$

При $x = 3,74$, $y = -0,825$, $z = 0,16$ значение $v = 1,0553$.

Вариант 4. $w = |\cos x - \cos y|^{(1+2\sin^2 y)} \left(1 + z + \frac{z^2}{2} + \frac{z^3}{3} + \frac{z^4}{4} \right).$

При $x = 0,4$, $y = -0,875$, $z = -0,475 \cdot 10^{-3}$ значение $w = 1,9873$.

Вариант 5. $\alpha = \ln \left(y^{-\sqrt{|x|}} \right) \left(x - \frac{y}{2} \right) + \sin^2 \operatorname{arctg}(z).$

При $x = -15,246$, $y = 4,642 \cdot 10^{-2}$, $z = 20,001 \cdot 10^{-2}$ значение $\alpha = -182,036$.

Вариант 6. $\beta = \sqrt{10(\sqrt[3]{x} + x^{y+2})} (\arcsin^2 z - |x - y|).$

При $x = 16,55 \cdot 10^{-3}$, $y = -2,75$, $z = 0,15$ значение $\beta = -38,902$.

Вариант 7. $\gamma = 5 \operatorname{arctg} x - \frac{x + 3|x - y| + x^2}{4(|x - y|z + x^2)} \arccos x.$

При $x = 0,1722$, $y = 6,33$, $z = 3,25 \cdot 10^{-4}$ значение $\gamma = -172,025$.

Вариант 8. $\varphi = \frac{e^{|x-y|}|x - y|^{x+y}}{\operatorname{arctg} x + \operatorname{arctg} z} + \sqrt[3]{x^6 + \ln^2 y}.$

При $x = -2,235 \cdot 10^{-2}$, $y = 2,23$, $z = 15,221$ значение $\varphi = 39,374$.

Вариант 9. $\psi = \left| x^{y/x} - \sqrt[3]{\frac{y}{x}} \right| + (y-x) \frac{\cos y - \frac{z}{y-x}}{1 + (y-x)^2}.$

При $x = 1,825 \cdot 10^2$, $y = 18,225$, $z = -3,298 \cdot 10^{-2}$ $\psi = 1,2131$.

Вариант 10. $a = 2^{-x} \sqrt{x + \sqrt[4]{|y|}} \sqrt[3]{e^{x-1/\sin z}}.$

При $x = 3,981 \cdot 10^{-2}$, $y = -1,625 \cdot 10^3$, $z = 0,512$ значение $a = 1,26185$.

Вариант 11. $b = y \sqrt[3]{|x|} + \frac{|x-y| \left(1 + \frac{\sin^2 z}{\sqrt{x+y}} \right)}{e^{|x-y|} + \frac{x}{2}} \cos^3 y.$

При $x = 6,251$, $y = 0,827$, $z = 25,001$ значение $b = 0,7121$.

Вариант 12. $c = 2^{y^x} + (3^x)^y - \frac{y \left(\operatorname{arctg} z - \frac{\pi}{6} \right)}{|x| + \frac{1}{y^2 + 1}}.$

При $x = 3,251$, $y = 0,325$, $z = 0,466 \cdot 10^{-4}$ значение $c = 4,025$.

Вариант 13. $f = \frac{\sqrt[4]{y + \sqrt[3]{x-1}}}{|x-y| (\sin^2 z + \operatorname{tg} z)}.$

При $x = 17,421$, $y = 10,365 \cdot 10^{-3}$, $z = 0,828 \cdot 10^5$ значение $f = 0,33056$.

Вариант 14. $g = \frac{y^{x+1}}{\sqrt[3]{|y-2|} + 3} + \frac{x + \frac{y}{2}}{2|x+y|} (x+1)^{-1/\sin z}.$

При $x = 12,3 \cdot 10^{-1}$, $y = 15,4$, $z = 0,252 \cdot 10^3$ значение $g = 82,8257$.

Вариант 15. $h = \frac{x^{y+1} + e^{y-1}}{1+x|y-\operatorname{tg} z|} (1+|y-x|) + \frac{|y-x|^2}{2} - \frac{|y-x|^3}{3}.$

При $x = 2,444$, $y = 0,869 \cdot 10^{-2}$, $z = -0,13 \cdot 10^3$ значение $h = -0,49871$.

Вариант 16. $f = \frac{\sin \pi x^2 + \ln y^2}{\sin \pi x^2 + \sin y + \ln z^2 + x^2 + e^{\cos y}}.$

При $x = 0,25$, $y = 4,64$, $z = 1,53$ значение $f = 3,126$.

Вариант 17. $g = \frac{\sqrt[4]{\ln(1+x)^2 + \cos(\pi y^3)}}{\left(\cos e^z + \sqrt{\frac{1}{x} + e^{y^2}}\right)^{\sin z}}.$

При $x = 3,46$, $y = 1,21$, $z = 0,69$ значение $g = 0,509$.

Вариант 18. $k = \sqrt[3]{\left(\sin x + y^2 + e^{\sin z}\right)^2 + \left(\ln z^2 - \sin \frac{\pi x^2}{4}\right)^3}.$

При $x = 1,73$, $y = 1,13$, $z = 0,91$ значение $k = 2,6788$.

Вариант 19. $m = \frac{\sqrt{\left(\cos x + z^3 + e^{\sin y}\right)^3 + \left(\ln x^2 - \sin \frac{\pi y^2}{4}\right)^2}}{\left(\ln(1+z^2) + \cos \pi x^3\right)^{\sin x} + \left(e^{y^2} + \cos e^z + \sqrt{1/x}\right)^{1/x}}.$

При $x = 2,3 \cdot 10^{-1}$, $y = 3,0 \cdot 10^{-3}$, $z = 8,9 \cdot 10^{-3}$ значение $m = 1,504 \cdot 10^{-2}$.

Вариант 20. $t = \frac{\left(e^{z^2} + \sqrt{1/x} + \cos e^y\right)^2}{\sqrt[3]{\cos \pi x^3 + \ln(1+y)^2}}.$

При $x = 5,62 \cdot 10^{-2}$, $y = 1,340 \cdot 10^3$, $z = 4,5 \cdot 10^{-1}$ значение $t = 10.5012$.

3. Программирование алгоритмов ветвления

3.1. Условные операторы

Операторы ветвления позволяют изменить порядок выполнения операторов в программе. К операторам ветвления относятся условный оператор `if` и оператор выбора `switch`.

Условный оператор `if` используется для разветвления процесса обработки данных на два направления. Он может иметь одну из форм: сокращённую или полную. Форма сокращённого оператора `if`:

```
if (B) S;
```

где B — логическое или арифметическое выражение, истинность которого проверяется; S — оператор. При выполнении сокращённой формы оператора `if` сначала вычисляется выражение B , затем проводится анализ его результата: если B истинно, то выполняется оператор S ; если B ложно, то оператор S пропускается. Таким образом, с помощью сокращённой формы оператора `if` можно либо выполнить оператор S , либо пропустить его.

Форма полного оператора `if` выглядит следующим образом:

```
if (B) S1; else S2;
```

где B — логическое или арифметическое выражение, истинность которого проверяется; $S1$, $S2$ — операторы. При выполнении полной формы оператора `if` сначала вычисляется выражение B , затем анализируется его результат: если B истинно, то выполняется оператор $S1$, а оператор $S2$ пропускается; если B ложно, то выполняется оператор $S2$, а $S1$ — пропускается. Таким образом, с помощью полной формы оператора `if` можно выбрать одно из двух альтернативных действий процесса обработки данных.

Для примера, вычислим значение функции:

$$y(x) = \begin{cases} \sin x, & x \leq a, \\ \cos x, & a < x < b, \\ \operatorname{tg} x, & x \geq b. \end{cases}$$

Указанное выражение может быть запрограммировано в виде

```
if (x <= a)
    y = Math.Sin(x);
if ((x > a) && (x < b))
    y = Math.Cos(x);
if (x >= b)
    y = Math.Sin(x) / Math.Cos(x);
```

или с помощью оператора `else`:

```
if (x <= a)
```

```

y = Math.Sin(x);
else
    if (x < b)
        y = Math.Cos(x);
    else
        y = Math.Sin(x) / Math.Cos(x);

```

Обратите внимание!

В C-подобных языках программирования, к которым относится и C#, операция сравнения представляется двумя знаками равенства, например: `if (a == b)`.

Оператор выбора **switch** предназначен для разветвления процесса вычислений по нескольким направлениям. Формат оператора (выражение, записанное в квадратных скобках, является необязательным элементом в операторе **switch**. Если оно отсутствует, то может отсутствовать и оператор перехода):

```

switch (<выражение>)
{
    case <константное_выражение_1>:
        [<оператор 1>];
        <оператор перехода>;
    case <константное_выражение_2>:
        [<оператор 2>];
        <оператор перехода>;
    ...
    case <константное_выражение_n>:
        [<оператор n>];
        <оператор перехода>;
    [default:
        <оператор>;]
}

```

Выражение, стоящее за ключевым словом **switch**, должно иметь арифметический, символьный или строковый тип. Все константные выражения должны иметь разные значения, но их тип должен совпадать с типом выражения, стоящим после **switch** или приводиться к нему. Ключевое слово **case** и расположенное после него константное выражение называют также *меткой case*.

Выполнение оператора начинается с вычисления выражения, расположенного за ключевым словом **switch**. Полученный результат сравнивается с меткой **case**. Если результат выражения соответствует метке **case**, то выполняется оператор, стоящий после этой метки, за которым обязательно должен следовать оператор перехода: **break**, **goto**, **return** и т. д. При использовании оператора **break** происходит выход из блока **switch** и управление передаётся оператору, следующему за блоком. Если же используется оператор **goto**, то управление передаётся оператору, помеченному меткой, стоящей после **goto**. Оператор **return** завершает выполнение текущего блока.

Если ни одно выражение `case` не совпадает со значением оператора `switch`, управление передаётся операторам, следующим за необязательным идентификатором `default`. Если `default` отсутствует, то управление передаётся за пределы оператора `switch`.

Пример использования оператора `switch` показан ниже.

```
int dayOfWeek = 5;
switch (dayOfWeek)
{
    case 1:
        MessageBox.Show("Понедельник");
        break;
    case 2:
        MessageBox.Show("Вторник");
        break;
    case 3:
        MessageBox.Show("Среда");
        break;
    case 4:
        MessageBox.Show("Четверг");
        break;
    case 5:
        MessageBox.Show("Пятница");
        break;
    case 6:
        MessageBox.Show("Суббота");
        break;
    case 7:
        MessageBox.Show("Воскресенье");
        break;
    default:
        MessageBox.Show("Неверное значение!");
        break;
}
```

Поскольку на момент выполнения оператора `switch` в этом примере переменная `dayOfWeek` равнялась 5, то будут выполнены операторы из блока `case 5`.

В ряде случаев оператор `switch` можно заменить несколькими операторами `if`, однако не всегда такая замена будет легче для чтения. Например, приведённый выше пример можно переписать так:


```
int dayOfWeek = 5;
if (dayOfWeek == 1)
    MessageBox.Show("Понедельник");
else
    if (dayOfWeek == 2)
        MessageBox.Show("Вторник");
    else
        if (dayOfWeek == 3)
            MessageBox.Show("Среда");
        else
            if (dayOfWeek == 4)
```

```

        MessageBox.Show("Четверг");
else
    if (dayOfWeek == 5)
        MessageBox.Show("Пятница");
    else
        if (dayOfWeek == 6)
            MessageBox.Show("Суббота");
        else
            if (dayOfWeek == 7)
                MessageBox.Show("Воскресенье");
            else
                MessageBox.Show("Неверное значение!");

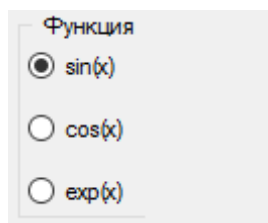
```

3.2. Элемент управления RadioButton

При создании программ в *MS Visual Studio* для организации разветвлений часто используются элементы управления  **RadioButton**.

Состояние такой кнопки («включено» – «выключено») визуально отражается на форме, а в программе можно узнать его с помощью проверки значения свойства **Checked**: если кнопка включена, это свойство будет содержать **True**, в противном случае — **False**.

Группируются радиокнопки с помощью какого-либо контейнера — часто это бывает элемент **GroupBox**. Если пользователь выбирает один из вариантов переключателя в группе, все остальные автоматически отключаются. Радиокнопки, размещённые в разных контейнерах, образуют независимые группы.



*Рис. 10 — Группа элементов **RadioButton**, расположенная в контейнере **GroupBox***

```

if (radioButton1.Checked)
    MessageBox.Show("Выбрана функция: синус");
else
    if (radioButton2.Checked)
        MessageBox.Show("Выбрана функция: косинус");
    else
        if (radioButton3.Checked)
            MessageBox.Show("Выбрана функция: экспонента");

```

3.3. Пример приложения для вычисления сложной функции

Задание: составить программу вычисления для заданных значений x , y , z функции

$$U(x, y, z) = \begin{cases} y \sin x + z, & z - x = 0, \\ y e^{\sin x} - z, & z - x < 0, \\ y \sin(\sin x) + z, & z - x > 0. \end{cases}$$

Вариант размещения элементов управления в окне приложения показан на рис. 11.

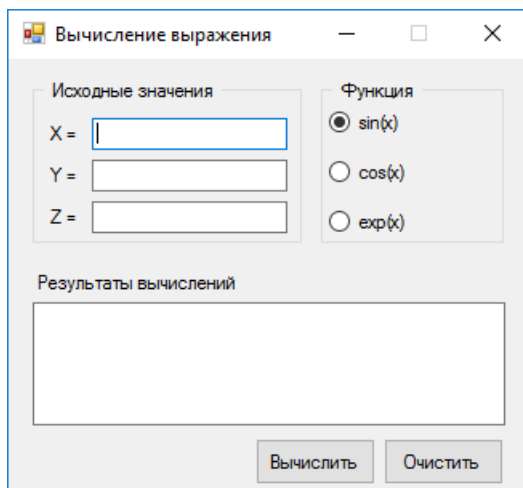


Рис. 11 — Вид приложения для вычисления сложной функции

Как и в предыдущем примере, для вывода результатов вычислений в приложении используется элемент управления `TextBox` со свойствами `Multiline = True` и `ScrollBars = Both`.

Таким же образом создаются и обработчики событий в приложении. Например, далее приведен фрагмент текста обработчика события нажатия на кнопку «Вычислить».

```
private void button1_Click(object sender, EventArgs e)
{
    // Получение исходных данных из текстовых полей
    double.Parse(textBox1.Text, out double x);
    ...

    // Вывод исходных данных в окно результатов
    textBox4.Text = "При X = " + textBox1.Text + Environment.NewLine;
    ...

    // Вычисление выражения
    double u = 0;
    if (radioButton1.Checked) {
        if ((z - x) == 0) {
            u = y * Math.Sin(x) * Math.Sin(x) + z;
        } else if ((z - x) < 0) {
            u = y * Math.Exp(Math.Sin(x)) - z;
        }
    }
}
```

```

    } else {
        u = y * Math.Sin(Math.Sin(x)) + z;
    }
} else if (radioButton2.Checked) {
    ...
} else if (radioButton3.Checked) {
    ...
}
// Вывод результата
textBox4.Text += "U = " + u.ToString("f2") + Environment.NewLine;
}

```

3.4. Выполнение индивидуального задания

По указанию преподавателя выберите индивидуальное задание из нижеприведенного списка. Отредактируйте вид формы и текст программы, в соответствии с полученным заданием. С помощью элемента управления `RadioButton` дайте пользователю возможность во время выполнения программы выбрать в качестве $f(x)$ следующие функции: $\operatorname{sh} x$, x^2 , e^x .

Вариант 1.

$$u = \begin{cases} (f(x) + y)^2 - \sqrt{y \cdot f(x)}, & xy > 0 \\ (f(x) + y)^2 + \sqrt{y \cdot f(x)}, & xy < 0 \\ (f(x) + y)^2 + 1, & xy = 0. \end{cases}$$

Вариант 4.

$$u = \begin{cases} (f(x) - y)^3 + \operatorname{arctg}(f(x)), & x > y, \\ (y - f(x))^3 + \operatorname{arctg}(f(x)), & y > x, \\ (y + f(x))^3 + 0,5, & y = x. \end{cases}$$

Вариант 2.

$$u = \begin{cases} \ln(f(x)) + (f(x)^2 + y)^3, & x/y > 0, \\ \ln\left|\frac{f(x)}{y}\right| + (f(x) + y)^3, & x/y < 0, \\ (f(x)^2 + y)^3, & x = 0, \\ 0, & y = 0. \end{cases}$$

Вариант 5.

$$u = \begin{cases} i\sqrt{f(x)}, & i - \text{чётное}, x > 0, \\ \frac{i}{2\sqrt{|f(x)|}}, & i - \text{нечётное}, x < 0, \\ \sqrt{|i \cdot f(x)|}, & \text{иначе.} \end{cases}$$

Вариант 6.

Вариант 3.

$$u = \begin{cases} f(x)^2 + y^2 + \sin y, & x - y = 0, \\ (f(x) - y)^2 + \cos y, & x - y > 0, \\ (y - f(x))^2 + \operatorname{tg} y, & x - y < 0. \end{cases}$$

$$u = \begin{cases} e^{f(x)-|b|}, & 0,5 < x \cdot b < 10, \\ \sqrt{|f(x) + b|}, & 0,1 < x \cdot b < 0,5, \\ 2f(x)^2, & \text{иначе.} \end{cases}$$

Вариант 7.

$$u = \begin{cases} e^{f(x)}, & 1 < x \cdot b < 10, \\ \sqrt{|f(x) + 4b|}, & 12 < x \cdot b < 40, \\ b(f(x))^2, & \text{иначе.} \end{cases}$$

Вариант 8.

$$u = \begin{cases} \sin(5f(x) + 3m|f(x)|), & -1 < m < x, \\ \cos(3f(x) + 5m|f(x)|), & x > m, \\ (f(x) + m)^2, & x = m. \end{cases}$$

Вариант 9.

$$u = \begin{cases} 2f(x)^3 + 3p^2, & x > |p| \\ |f(x) - p|, & 3 < x < |p| \\ (f(x) - p)^2, & x = |p|. \end{cases}$$

Вариант 10.

$$u = \begin{cases} \ln(|f(x)| + |q|), & |x \cdot q| > 10, \\ e^{f(x)+q}, & |x \cdot q| < 10, \\ f(x) + q, & |x \cdot q| = 10. \end{cases}$$

Вариант 11.

$$u = \frac{\max(f(x), y, z)}{\min(f(x), y)} + 5.$$

Вариант 12.

$$u = \frac{\min(f(x) + y, y - z)}{\max(f(x), y)}.$$

Вариант 13.

$$u = \frac{|\min(f(x), y) - \max(y, z)|}{2}.$$

Вариант 14.

$$u = \frac{\max(f(x) + y + z, x \cdot y \cdot z)}{\min(f(x) + y + z, x \cdot y \cdot z)}.$$

Вариант 15.

$$u = \max(\min(f(x), y), z).$$

Вариант 16.

$$u = \begin{cases} (f(x) - y)^3 + \operatorname{tg}(f(x) + y), & x > y, \\ (y - f(x))^2 + \sin(f(x)), & y > x, \\ (y + f(x))^3 + \frac{3}{4}, & y = x. \end{cases}$$

Вариант 17.

$$u = \begin{cases} 2(f(x))^3 - 4p^2, & x > |p|, \\ |f(x) + px|, & 3 < x < |p|, \\ \left(\frac{f(x)}{p}\right)^2, & x = |p|. \end{cases}$$

Вариант 18.

$$u = \frac{\max\left(f(x) - \frac{y+z}{x}, x \cdot y \cdot z\right)}{\min\left(f(x) - \frac{y+z}{x}, x \cdot y \cdot z\right)}.$$

Вариант 19.

$$u = \begin{cases} 3\sqrt{f(x) + 3} - \frac{g}{2x}, & x < 0,5, \\ \sin(f(x) + 2p), & 0,5 \leq x \leq |p|, \\ \left(\frac{f(x) - p}{p^2 - 2p + f(x)^2}\right)^3, & x > |p|. \end{cases}$$

Вариант 20.

$$u = \begin{cases} \sin(f(x)^2 + 3x), & x > \pi, \\ \cos(4x^2 + xf(x) - 7), & x < \pi, \\ \operatorname{tg} f(x), & -\frac{\pi}{2} \leq f(x) \leq \frac{\pi}{2}. \end{cases}$$

Список использованных источников

1. Троелсен Э. С# и платформа .NET / [пер. с англ. Р. Михеев]. — Санкт-Петербург [и др.]: Питер, 2007. — 796 с.
2. Рихтер Дж. CLR via C#. Программирование на платформе Microsoft.NET Framework 2.0 на языке C#: [пер. с англ.]. — Санкт-Петербург [и др.]: Питер, 2007. — 636 с.
3. Павловская, Т. А. С#. Программирование на языке высокого уровня. Учебник для вузов — СПб.: Питер, 2007. — 432 с.
4. Марченко А. Л. Основы программирования на C# 2.0: учебное пособие. — Москва: Интернет-Университет Информационных Технологий: БИНОМ. Лаборатория знаний, 2007. — 551 с.
5. Нортроп Т. Основы разработки приложений на платформе Microsoft .NET Framework: учебный курс Microsoft / [пер. с англ. под общ. ред. А. Е. Соловченко]. — Санкт-Петербург [и др.]: Питер, 2007. — 842 с.
6. Дёмин А. Ю. Лабораторный практикум по информатике: учебное пособие / А. Ю. Дёмин, В. А. Дорофеев; Томский политехнический университет. — Томск: Изд-во Томского политехнического университета, 2014. — 134 с.
7. Потапова, Л. Е. Алгоритмизация и программирование на языке C#: метод. рекомендации к выполнению лабораторных работ / Л. Е. Потапова, Т. Г. Алейникова. — Витебск: ВГУ имени П. М. Машерова, 2014. — 50 с.

