

МИНИСТЕРСТВО НАУКИ И ВЫСШЕГО ОБРАЗОВАНИЯ РОССИЙСКОЙ ФЕДЕРАЦИИ
ФЕДЕРАЛЬНОЕ ГОСУДАРСТВЕННОЕ АВТОНОМНОЕ ОБРАЗОВАТЕЛЬНОЕ
УЧРЕЖДЕНИЕ ВЫСШЕГО ОБРАЗОВАНИЯ
«СЕВЕРО-КАВКАЗСКИЙ ФЕДЕРАЛЬНЫЙ УНИВЕРСИТЕТ»

Методические указания

по выполнению лабораторных работ
по дисциплине «Информационные технологии и программирование»
для студентов
Специальности 10.05.03 Информационная безопасность
автоматизированных систем
Специализация «Анализ безопасности информационных систем»

Ставрополь
2026

Содержание

ВВЕДЕНИЕ	3
Лабораторная работа № 1. Изучение среды разработки VISUAL STUDIO.....	4
ЛАБОРАТОРНАЯ РАБОТА №2. Программирование линейных алгоритмов	15
ЛАБОРАТОРНАЯ РАБОТА №3. Программирование разветвляющихся алгоритмов	25
ЛАБОРАТОРНАЯ РАБОТА №4. Программирование циклических алгоритмов	33
ЛАБОРАТОРНАЯ РАБОТА № 5. Программирование с использованием строк	40
ЛАБОРАТОРНАЯ РАБОТА № 6. Программирование с использованием одномерных массивов	44
ЛАБОРАТОРНАЯ РАБОТА № 7. Программирование с использованием многомерных массивов	51
ЛАБОРАТОРНАЯ РАБОТА № 8. Программирование с использованием средств для отображения графической информации	56
ЛАБОРАТОРНАЯ РАБОТА № 9. Программирование графики.....	62
ЛАБОРАТОРНАЯ РАБОТА № 10. Простейшая анимация	69
ЛАБОРАТОРНАЯ РАБОТА № 11. Обработка изображений	73
ЛАБОРАТОРНАЯ РАБОТА № 12. Методы	75
ЛАБОРАТОРНАЯ РАБОТА № 13. Сортировка.....	78
Список рекомендуемых источников.....	84

ВВЕДЕНИЕ

Цель и задачи освоения дисциплины

Целью освоения дисциплины «Информационные технологии и программирование» является формирование у будущего специалиста по специальности 10.05.03 Информационная безопасность автоматизированных систем специализация «Анализ безопасности информационных систем» заключается в формировании у будущих специалистов ключевых компетенций в области программирования и ИТ, необходимых для анализа, проектирования и обеспечения безопасности автоматизированных систем.

Задачи дисциплины:

- Формирование навыков программирования
- Освоение современных информационных технологий
- Анализ безопасности ПО

Лабораторная работа № 1. Изучение среды разработки VISUAL STUDIO

Цель лабораторной работы: изучить среду быстрой разработки приложений Visual Studio. Научится размещать и настраивать внешний вид элементов управления на форме.

Интегрированная среда разработчика Visual Studio

Среда 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.

При запуске Visual Studio появляется начальная страница со списком последних проектов, а также командами «Создать проект...» и «Открыть проект...». Нажмите ссылку «Создать проект...» или выберите в меню Файл команду «Создать проект...», на экране появится диалог для создания нового проекта (рис. 1.1).

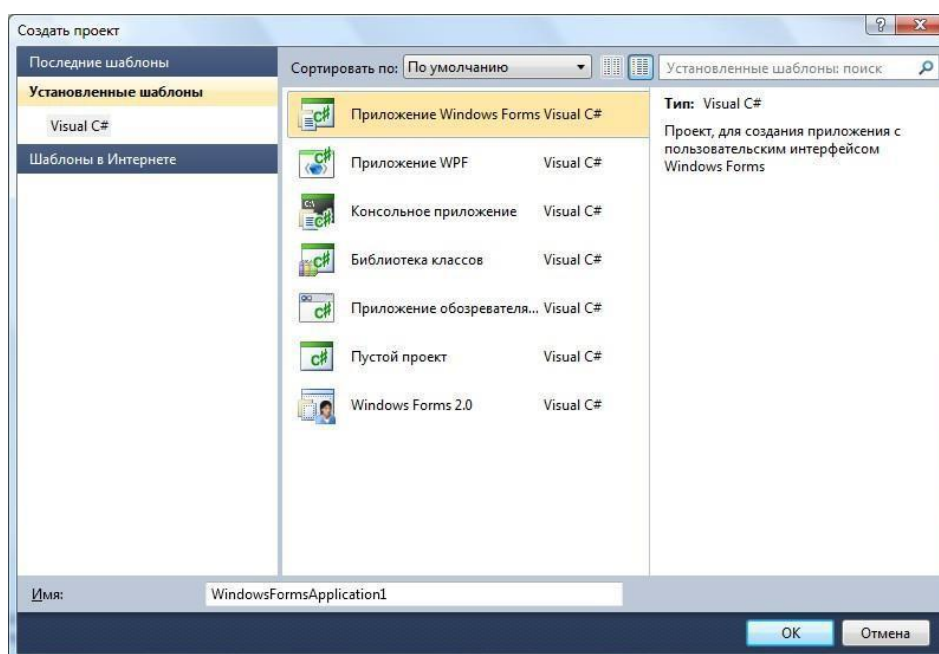


Рис 1.1. Диалог создания нового проекта.

Слева в списке шаблонов приведены языки программирования, которые поддерживает данная версия Visual Studio: убедитесь, что там выделен раздел Visual C#. В средней части приведены типы проектов, которые можно создать. В

наших лабораторных работах будут использоваться два типа проектов:

- Приложение Windows Forms – 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 – 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.

Выберите в списке тип проекта «Приложение Windows Forms», в поле «имя» внизу окна введите желаемое имя проекта (например, MyFirstApp) и нажмите кнопку ОК. Через несколько секунд Visual Studio создаст проект, и Вы сможете увидеть на экране картинку, подобную представленной на рис. 1.2.

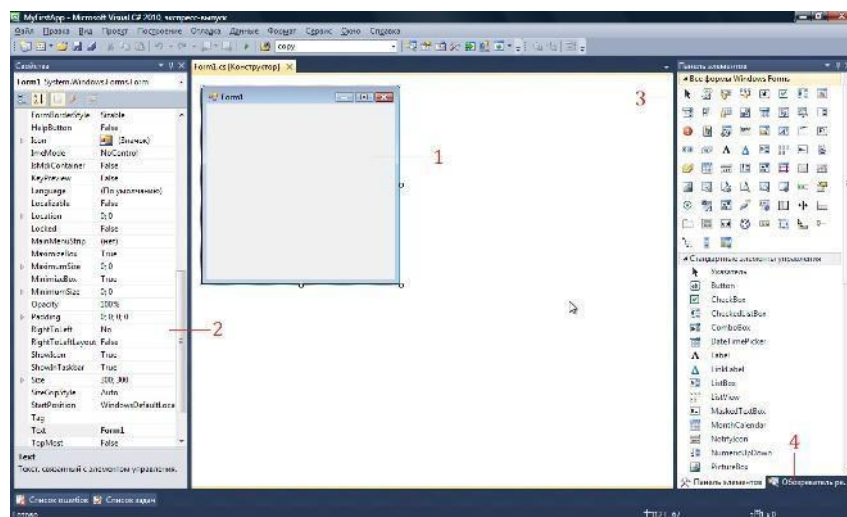


Рис 1.2. Главное окно Visual Studio

В главном окне Visual Studio присутствует несколько основных элементов, которые будут помогать нам в работе. Прежде всего, это **форма**

(1) – будущее окно нашего приложения, на котором будут размещаться элементы управления. При выполнении программы помещенные элементы управления будут иметь тот же вид, что и на этапе проектирования.

Второй по важности объект – это окно свойств (2), в котором приведены

все основные свойства выделенного элемента управления или окна. С помощью кнопки  можно просматривать свойства элемента управления, а кнопка переключает окно в режим просмотра событий. Если этого окна на экране нет, его можно активировать в меню Вид -> Окно свойств.

Сами элементы управления можно брать на 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.

Наконец, **обозреватель решений** (4) содержит список всех файлов, входящих в проект, включая добавленные изображения и служебные файлы. Активируется командой Вид -> Обзорщик решений.

Окно текста программы предназначено для просмотра, написания и редактирования текста программы. Переключаться между формой и текстом программы можно с помощью команд Вид -> Код (F7) и Вид -> Конструктор (Shift+F7). При первоначальной загрузке в окне текста программы находится текст, содержащий минимальный набор операторов для нормального функционирования пустой формы в качестве Windows-окна. При помещении элемента управления в окно формы, текст программы автоматически дополняется описанием необходимых для его работы библиотек стандартных программ (раздел `using`) и переменных для доступа к элементу управления (в скрытой части класса формы).

Программа на языке C# составляется как описание алгоритмов, которые необходимо выполнить, если возникает определенное событие, связанное с формой (например щелчок «мыши» на кнопке – событие Click, загрузка формы – Load). Для каждого обрабатываемого в форме события, с помощью окна свойств, в тексте программы организуется метод, в котором программист записывает на языке C# требуемый алгоритм.

Настройка формы

Настройка формы начинается с настройки размера формы. С помощью

мыши, «захватывая» одну из кромок формы или выделенную строку заголовка отрегулируйте нужные размеры формы.

Для настройки будущего окна приложения задаются свойства формы. Для задания любых свойств формы и элементов управления на форме используется окно свойств.

Новая форма имеет одинаковые имя (Name) и заголовок (Text) - Form1. Для изменения заголовка перейдите в окно свойств и щелкните кнопкой мыши на форме. В форме инспектора объектов найдите и щелкните мышью на строчке с названием Text. В выделенном окне наберите — Лаб. раб. N1. Ст. гр. 7А62 Иванов А.А.». Для задания цвета окна используйте свойство BackColor.

Размещение элементов управления на форме

Для размещения различных элементов управления на форме используется панель элементов. Панель элементов содержит элементы управления, сгруппированные по типу. Каждую группу элементов управления можно свернуть, если она в настоящий момент не нужна. Для выполнения лабораторных работ потребуются элементы управления из группы Стандартные элементы управления.

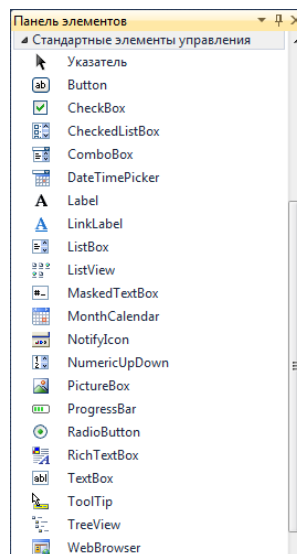


Рис.1.3 Панель элементов

Щёлкните на нужном элементе управления, а затем щёлкните в нужном месте формы — элемент появится на форме. Элемент можно перемещать по форме схватившись за него левой кнопкой мышки (иногда это можно сделать

лишь за появляющийся при нажатии на элемент квадрат со стрелками ). Если элемент управления позволяет изменять размеры, то на соответствующих его сторонах появятся белые кружки, ухватившись за которые и можно изменить размер. После размещения элемента управления на форме, его можно выделить щелчком мыши и при этом получить доступ к его свойствам в окне свойств.

Размещение строки ввода (TextBox)

Если необходимо ввести из формы в программу или вывести на форму информацию, которая вмещается в одну строку, используют окно однострочного редактора текста, представляемого элементом управления TextBox.

В данной программе с помощью однострочного редактора будут вводиться переменные `x`, `y`, `z` типа `double` или `int`.

Выберите на панели элементов пиктограмму , щелкните мышью в том месте формы, где вы хотите ее поставить. Вставьте три элемента TextBox в форму. Захватывая их —мышью— отрегулируйте размеры окон и их положение. Обратите внимание на то, что теперь в тексте программы можно использовать переменные `textBox1`, `textBox2` и `textBox3`, которые соответствуют каждому добавленному элементу управления. В каждой из этих переменных в свойстве `Text` будет содержаться строка символов (тип `string`) и отображаться в соответствующем окне TextBox.

С помощью инспектора объектов установите шрифт и размер символов, отражаемых в строке TextBox (свойство `Font`).

Размещение надписей (Label)

На форме могут размещаться пояснительные надписи. Для нанесения таких надписей на форму используется элемент управления Label. Выберите на панели элементов пиктограмму, щелкните на ней мышью. После этого в нужном месте формы щелкните мышью, появится надпись `label1`. Прodelайте это для четырех надписей. Для каждой надписи, щелкнув на ней мышью, отрегулируйте размер и, изменив свойство `Text` в окне свойств, введите строку, например —Введите значение `X`!;, а также выберите размер символов (свойство `Font`).

Обратите внимание, что в тексте программы теперь можно обращаться к четырём новым переменным типа Label. В них хранятся пояснительные строки, которые можно изменять в процессе работы программы.

Написание программы обработки события

С каждым элементом управления на форме и с самой формой могут происходить события во время работы программы. Например, с кнопкой может произойти событие – нажатие кнопки, а с окном, которое проектируется с помощью формы, может произойти ряд событий: создание окна, изменение размера окна, щелчок мыши на окне и т.п. Эти события могут быть обрабатываться в программе. Для обработки таких событий необходимо создать обработчики события – специальный метод. 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 кнопки таким образом создаётся обработчик события нажатия.

Написание программы обработки события нажатия кнопки (Click)

Поместите на форму кнопку, которая описывается элементом управления Button, для чего выберем пиктограмму. С помощью окна свойств измените заголовок (Text) на слово —Выполнить или другое по вашему желанию. Отрегулируйте положение и размер кнопки.

После этого два раза щелкните мышью на кнопке, появится текст программы:

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

Это и есть обработчики события нажатия кнопки. Вы можете добавлять свой код между скобками {}. Например, наберите:

```
MessageBox.Show("Привет!");
```

Написание программы обработки события загрузки формы (Load)

Второй способ создания обработчика события заключается в выборе соответствующего события для выделенного элемента на форме. При этом используется окно свойств и его закладка. Рассмотрим этот способ. Перейдите

на форму, в окне свойств найдите событие **Load**. Щелкните по данной строчке дважды мышкой. Появится метод:

```
private void Form1_Load(object sender, EventArgs)
```

Между скобками **{ }** вставим текст программы:

```
BackColor = Color. AntiqueWhite;
```

Каждый элемент управления имеет свой набор обработчиков событий, однако некоторые из них присущи большинству элементов управления. Наиболее часто применяемые события представлены в 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 e.KeyChar имеет тип char и содержит код нажатой клавиши (клавиша Enter клавиатуры имеет код #13, клавиша Esc - #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 Shift, Alt и Ctrl, а также о нажатой кнопке мыши. Информация о клавише передается параметром e.KeyCode, который представляет собой перечисление Keys с кодами всех клавиш, а информацию о клавишах-модификаторах Shift и др. можно узнать из параметра e.Modifiers
KeyUp	Является парным событием для OnKeyDown 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

Запуск и работа с программой

Запустить программу можно выбрав в меню **Отладка** команду **Начать отладку**. При этом происходит трансляция и, если нет ошибок, компоновка программы и создание единого загружаемого файла с расширением .exe. На экране появляется активное окно программы. Для завершения работы программы и возвращения в режим проектирования формы не забудьте закрыть окно программы!

Динамическое изменение свойств

Свойства элементов на окне 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";
```

<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 замены в окне свойств в свойстве (**Name**) для текущего элемента. Допускается использование латинских или русских символов, знака подчеркивания и цифр (цифра не должна стоять в начале идентификатора). Список свойств для конкретного элемента можно посмотреть в окне свойств, а также в приложении к данным методическим указаниям.

Выполнение индивидуального задания

Ниже приведено 15 вариантов задач. По указанию преподавателя выберите свое индивидуальное задание. Уточните условие задания, количество, наименование, типы исходных данных. Прочтите в приложении описание свойств и описание элементов управления Form, Label, TextBox, Button. С помощью окна свойств установите первоначальный цвет формы, шрифт выводимых символов.

Индивидуальные задания

Разместите на форме четыре кнопки (Button). Сделайте на кнопках следующие надписи: красны, зеленый, синий, желтый. Создайте четыре обработчика события нажатия на данные кнопки, которые будут менять цвет формы в соответствии с текстом на кнопках.

Разместите на форме две кнопки (Button) и одну метку (Label). Сделайте на кнопках следующие надписи: привет, до свидания. Создайте обработчики события нажатия на данные кнопки, которые будут менять текст метки, на слова: привет, до свидания. Создайте обработчик события создания формы (Load), который будет устанавливать цвет формы и менять текст метки на строку «Начало работы».

Разместите на форме две кнопки (Button) и одну метку (Label). Сделайте на кнопках следующие надписи: скрыть, показать. Создайте обработчики события нажатия на данные кнопки, которые будут скрывать или показывать метку. Создайте обработчик события создания формы (Load), который будет устанавливать цвет формы и менять текст метки на строку

«Начало работы».

Разместите на форме три кнопки (Button) и одно поле ввода (TextBox). Сделайте на кнопках следующие надписи: скрыть, показать, очистить. Создайте обработчики события нажатия на данные кнопки, которые будут скрывать или показывать поле ввода. При нажатии на кнопку

«очистить» текст из поля ввода должен быть удален.

Разместите на форме две кнопки (Button) и одно поле ввода (TextBox). Сделайте на кнопках следующие надписи: заполнить, очистить. Создайте обработчики события нажатия на данные кнопки, которые будут очищать или заполнять поле ввода знаками «*****». Создайте обработчик события создания формы (Load), который будет устанавливать цвет формы и менять текст в поле ввода на строку «+++++».

Разработайте игру, которая заключается в следующем. На форме размещены пять кнопок (Button). При нажатии на кнопку какие-то кнопки становятся видимыми, а какие-то невидимыми. Цель игры скрыть все кнопки.

Разработайте игру, которая заключается в следующем. На форме размещены четыре кнопки (Button) и четыре метки (Label). При нажатии на кнопку часть надписей становится невидимыми, а часть, наоборот, становятся

видимыми. Цель игры скрыть все надписи.

Разместите на форме ряд кнопок (Button). Создайте обработчики события нажатия на данные кнопки, которые будут делать неактивными текущую кнопку. Создайте обработчик события изменение размера формы (Resize), который будет устанавливать все кнопки в активный режим.

Разместите на форме ряд кнопок (Button). Создайте обработчики события нажатия на данные кнопки, которые будут делать неактивными следующую кнопку. Создайте обработчик события нажатия кнопки мыши на форме (Click), который будет устанавливать все кнопки в активный режим.

Разместите на форме три кнопки (Button) и одно поле ввода (TextBox). Сделайте на кнопках следующие надписи: *****, +++++, 00000. Создайте обработчики события нажатия на данные кнопки, которые будут выводить текст, написанный на кнопках, в поле ввода. Создайте обработчик события создания формы (Load), который будет устанавливать цвет формы и менять текст в поле ввода на строку «Готов к работе».

Разместите на форме ряд полей ввода (TextBox). Создайте обработчики события нажатия кнопкой мыши на данные поля ввода, которые будут выводить в текущее поле ввода его номер. Создайте обработчик события изменение размера формы (Resize), который будет очищать все поля ввода.

Разместите на форме поле ввода (TextBox), метку (Label) и кнопку (Button). Создайте обработчик события нажатия на кнопку, который будет копировать текст из поля ввода в метку. Создайте обработчик события нажатия кнопки мыши на форме (Click), который будет устанавливать цвет формы и менять текст метки на строку «Начало работы» и очищать поле ввода.

Разместите на форме поле ввода (TextBox), и две кнопки (Button) с надписями: заблокировать, разблокировать. Создайте обработчики события нажатия на кнопки, которые будут делать активным или неактивным поле ввода. Создайте обработчик события нажатия кнопки мыши на форме (Click), который будет устанавливать цвет формы и делать невидимыми все элементы.

Реализуйте игру минер на поле 3x3 из кнопок (Button). Первоначально все кнопки не содержат надписей. При попытке нажатия на кнопку на ней либо показывается количество мин, либо надпись «мина!» и меняется цвет окна.

Разместите на форме четыре кнопки (Button). Напишите для каждой обработчик события, который будет менять размеры и местоположение на окне других кнопок.

ЛАБОРАТОРНАЯ РАБОТА №2. Программирование линейных алгоритмов

Цель лабораторной работы: научиться составлять каркас простейшей программы в среде Visual Studio. Написать и отладить программу линейного алгоритма.

Структура приложения

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 как стартовый проект. Выполнение решения начинается со стартового проекта.

Таким образом, при создании простейшей C# программы в Visual Studio. Создается папка решения, в которой для каждого проекта создается подпапка проекта, в которой будут создаваться другие подпапки с результатами компиляции приложения.

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

В предыдущей лабораторной работе мы попробовали создавать оконные приложения или иначе **Приложения Windows Forms**. Как пример другого типа проекта можно привести проект *консольного* приложения.

По своим "внешним" проявлениям консольные 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.

Проект в Visual Studio состоит из файла проекта (файл с расширением *.csproj*), одного или нескольких файлов исходного текста (с расширением *.cs*), файлов с описанием окон формы (с расширением *.designer.cs*), файлов ресурсов (с расширением *.resx*), а также ряда служебных файлах.

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

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

```
// Раздел подключенных пространств имен
using System;
// Пространство имен нашего проекта
namespace MyFirstApp
{
    // Класс окна
    public partial class Form1 : Form
    {
        // Методы окна
        public Form1()
        {
            InitializeComponent();
        }
    }
}
```

В разделе подключения пространств имен (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.

Для того чтобы не происходило конфликтов имён классов и переменных, классы нашего проекта также помещаются в отдельное пространство имен. Определяется оно ключевым словом `namespace`, после которого следует имя пространства (обычно оно совпадает с именем проекта).

Внутри пространства имен помещаются наши классы – в новом проекте это класс окна, который содержит все методы для управления поведением окна. Обратите внимание, что в определении класса присутствует ключевое слово `partial`, это говорит о том, что в исходном тексте представлена только часть класса, с которой мы работаем непосредственно, а служебные методы для обслуживания окна скрыты в другом модуле (при желании их тоже можно посмотреть, но редактировать вручную не рекомендуется).

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

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

Работа с проектом

Как вы видите, проект в Visual Studio состоит из многих файлов, и создание сложной программы требует хранения каждого проекта в отдельной папке. При создании нового проекта Visual Studio по умолчанию сохраняет его в отдельной папке. Рекомендуется создать для себя свою папку со своей фамилией внутри папки своей группы, чтобы все проекты хранились в одном месте. После этого можно запускать Visual Studio и создавать новый проект (как это сделать показано в предыдущей лабораторной работе).

Сразу после создания проекта рекомендуется сохранить его в подготовленной папке: **Файл -> Сохранить всё**. При внесении значительных изменений в проект следует еще 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 *.csproj* и сделать на нём двойной щелчок.

Описание данных

Типы данных имеют особенное значение в С#, поскольку это строго типизированный язык. Это означает, что все операции подвергаются строгому контролю со стороны компилятора на соответствие типов, причем недопустимые операции не компилируются. Такая строгая проверка типов позволяет предотвратить ошибки и повысить надежность программ. Для обеспечения контроля типов все переменные, выражения и значения должны принадлежать к определенному типу. Такого понятия, как "бестиповая" переменная, в данном языке программирования вообще не существует. Более того, тип значения определяет те операции, которые разрешается выполнять над ним. Операция, разрешенная для одного типа данных, может оказаться недопустимой для другого.

В С# имеются две общие категории встроенных типов данных: типы значений и ссылочные типы. Они отличаются по содержимому переменной. Концептуально разница между ними состоит в том, что тип значения (value type) хранит данные непосредственно, в то время как ссылочный тип (reference type) хранит ссылку на значение.

Эти типы сохраняются в разных местах памяти: типы значений сохраняются в области, известной как стек, а ссылочные типы — в области, называемой управляемой кучей.

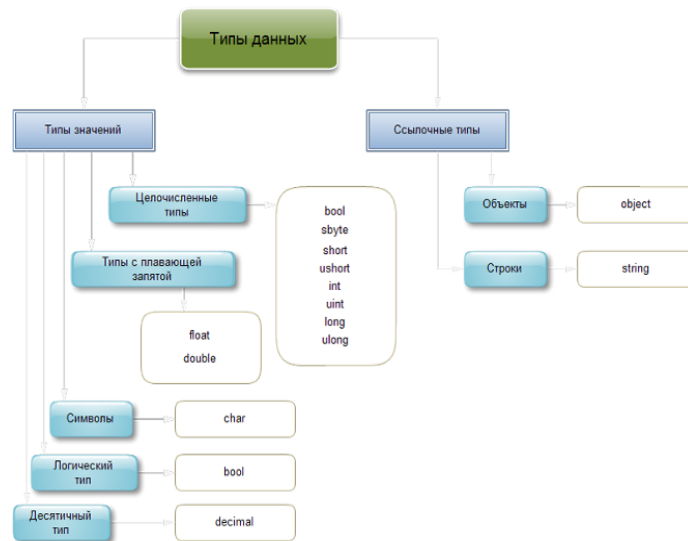


Рис.2.1 Панель элементов

- **Целочисленные типы**

В C# определены девять целочисленных типов: char, byte, sbyte, short, ushort, int, uint, long и ulong. Но тип char в основном применяется для представления символов и поэтому рассматривается отдельно. Остальные восемь целочисленных типов предназначены для числовых расчетов.

- **Типы с плавающей точкой**

Типы с плавающей точкой позволяют представлять числа с дробной частью. В C# имеются две разновидности типов данных с плавающей точкой: float и double. Они представляют числовые значения с одинарной и двойной точностью соответственно.

- **Десятичный тип данных**

Для представления чисел с плавающей точкой высокой точности предусмотрен также десятичный тип decimal, который предназначен для применения в финансовых вычислениях.

- **Символы**

В C# символы представлены не 8-разрядным кодом, как во многих других языках программирования, например C++, а 16-разрядным кодом, который называется юникодом (Unicode). В юникоде набор символов представлен настолько широко, что он охватывает символы практически из всех естественных языков на свете.

- **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.

- Строки

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. Класс `String` в языке `C#` относится к ссылочным типам. Над строками - объектами этого класса - определен широкий набор операций, соответствующий современному представлению о том, как должен быть устроен строковый тип. По сути, текст хранится в виде последовательной доступной только для чтения коллекции объектов `Char`.

Рассмотрим самые популярные данные – переменные и константы. Переменная - это ячейка памяти, которой присвоено некоторое имя и это имя используется для доступа к данным, расположенным в данной ячейке. Для каждой переменной задаётся тип данных – диапазон всех возможных значений для данной переменной. Объявляются переменные непосредственно в тексте программы. Лучше всего сразу присвоить им начальное значение с помощью знака присвоения `"="` (*переменная = значение*):

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

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 значением
```

Несмотря на то, что тип `char` определен в `C#` как целочисленный, его не следует путать со всеми остальными целочисленными типами.

Частным случаем переменных являются константы. Константы - это переменные, значения которых не меняются в процессе выполнения программы. Константы описываются как обычная переменная, только с ключевым словом `const` впереди:

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

Ввод/вывод данных в программу

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

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

Однако со строкой символов трудно производить арифметические операции, поэтому лучше всего при вводе числовых данных перевести строку в целое или вещественное число. Для этого у типов, или `int` и `double` существуют методы `Parse` для преобразования строк в числа. С этими числами можно производить различные арифметические действия. Таким образом, предыдущий пример можно переделать следующим образом:

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

Перед выводом числовые данные следует преобразовать назад в строку. Для этого у каждой переменной существует метод `ToString`, который возвращает в результате строку с символьным представлением значения. Вывод данных можно осуществлять в элементы **TextBox** или **Label**, используя свойство **Text**. Например:

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

Арифметические действия и стандартные функции

При вычислении выражения стоящего в правой части оператора

присвоения могут использоваться арифметические операции: * умножение, + сложение, - вычитание, / деление, % взятие остатка при делении. Для задания приоритетов операций могут использоваться круглые скобки (). Также могут использоваться стандартные математические функции, представленные методами класса Math:

- Math.Sin(a) – синус (аргумент задается в радианах);
- Math.Cos(a) – косинус (аргумент задается в радианах);
- Math.Atan(a) – арктангенс (аргумент задается в радианах);
- Math.Log(a) – натуральный логарифм;
- Math.Exp(a) – экспонента;
- Math.Pow(x,y) – возводит переменную x в степень y;
- Math.Sqrt(a) – квадратный корень;
- Math.Abs(a) – модуль числа;
- Math.Truncate(a) – целая часть числа;
- Math.Round(a) – округление числа;

Более подробную информацию смотрите в приложении.

Для вывода результатов работы программы в программе используется текстовое окно, которое представлено обычным элементом управления. После установки свойства **Multiline** в **True** появляется возможность растягивать элемент управления не только по горизонтали, но и по вертикали. А после установки свойства **ScrollBars** в значение **Both** в окне 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 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 окна добавляется символ

перевода курсора на новую строку (который может отличаться в разных операционных системах и потому представлен свойством класса `Environment`) и сама новая строка. Если добавляется числовое значение, то его предварительно нужно привести в символьный вид методом `ToString()`.

Работа с программой происходит следующим образом. Нажмите (щелкните мышью) кнопку —Выполнить». В окне `textBox4` появляется результат. Измените исходные значения `x`, `y`, `z` в окнах `textBox1` – `textBox3` и снова нажмите кнопку «Выполнить» - появятся новые результаты.

Полный текст программы имеет следующий вид:

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

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

        private void Form1_Load(object sender, EventArgs e)
        {
            A.A.";
            textBox1.Text = "3,4";
            // Начальное значение
            X    textBox2.Text    =
            "0,74"; // Начальное
            значение                Y
            textBox3.Text          =
            "19,43"; // Начальное
            значение Z
            // Вывод строки в
            многострочный редактор
            textBox4.Text = "Лаб. раб. N1.
            Ст. гр. 7А62 Иванов
        }
```

```

        {
            // Считывание значения X
            double x =
            double.Parse(textBox1.Text);

            // Вывод значения X в окно
            textBox4.Text +=
                Environment.NewLine
                + "X = " +
                x.ToString();

            // Считывание значения Y
            double y =
            double.Parse(textBox2.Text);

            // Вывод значения Y в окно
            textBox4.Text +=
                Environment.NewLine
                + "Y = " +
                y.ToString();

            // Считывание значения Z
            double z =
            double.Parse(textBox3.Text);

            // Вывод значения Z в окно
            textBox4.Text +=
                Environment.NewLine
                + "Z = " +
                z.ToString();

            // Вычисляем арифметическое
            // выражение
            double a = Math.Tan(x + y)
            * Math.Tan(x + y); double b
            = Math.Exp(y - z);

            double c = Math.Sqrt(Math.Cos(x *
            x) + Math.Sin(z *

            double u = a - b * c;

            // Выводим результат в окно
            textBox4.Text +=
                Environment.NewLine +
                "Результат U = " + u.ToString();
        }
    }
}

```

ЛАБОРАТОРНАЯ РАБОТА №3. Программирование разветвляющихся алгоритмов

Цель лабораторной работы: научиться пользоваться простейшими компонентами организации переключений (RadioButton). Написать и отладить программу разветвляющегося алгоритма.

Логические переменные и операции над ними

Переменные логического типа описываются посредством служебного слова **bool**. Они могут принимать только два значения - **False** (ложь) и **True** (истина). Результат **False** (ложь) и **True** (истина) возникает при использовании операций сравнения **>** меньше, **<** больше, **!=** не равно, **>=** меньше или равно, **<=** больше или равно, **==** равно. Описываются логические переменные можно так:

bool b;

В языке C# имеются логические операции, применяемые к переменным логического типа. Это операции логического отрицания (**!**), логическое И (**&&**) и логическое ИЛИ (**||**). Операция логического отрицания является унарной операцией. Результат операции **!** есть **False**, если операнд истинен, и **True**, если операнд имеет значение ложь. Так,

! True → **False** (неправда есть ложь)

! False → **True** (не ложь есть правда)

Результат операции логическое И (**&&**) есть истина, только если оба ее операнда истинны, и ложь во всех других случаях. Результат операции логическое ИЛИ (**||**) есть истина, если какой-либо из ее операндов истинен, и ложь только тогда, когда оба операнда ложны. 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: сокращенную или

полную.

Форма сокращенного оператора **if**:

if (B) S;

где **B** - логическое или арифметическое выражение, истинность которого проверяется; **S** - оператор: простой или составной.

При выполнении сокращенной 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 сокращенной формы оператора **if** можно либо выполнить оператор **S**, либо пропустить его.

Форма полного оператора **if**:

if (B) S1; else S2;

где **B** - логическое или арифметическое выражение, истинность которого проверяется; **S1, S2** - оператор: простой или составной.

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.

Пример. Вычислим значение функции

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

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

if (x <= a) y = Math.Sin(x);

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

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

Оператор выбора **switch** предназначен для разветвления процесса

вычислений по нескольким направлениям. Формат оператора:

```
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>; ]  
}
```

Замечание. Выражение, записанное в квадратных скобках, является необязательным элементом в операторе **switch**. Если оно отсутствует, то может отсутствовать и оператор перехода. Выражение, стоящее за ключевым словом **switch**, должно иметь арифметический, символьный, строковый тип или тип указатель. 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** и т.д. При использовании оператора **break** происходит выход из **switch** и управление передается оператору, следующему за **switch**. Если же используется оператор **goto**, то управление передается оператору, помеченному меткой, стоящей после **goto**.

Если ни одно выражение **case** не совпадает со значением оператора **switch**, управление передается операторам, следующим за необязательной подписью

default. Если подписи **default** нет, то управление передается за пределы оператора **switch**.

Пример использования оператора **switch: int**

```
caseSwitch = 1;
switch (caseSwitch)
{
case 1:
Console.WriteLine("Case 1"); break;
case 2:
Console.WriteLine("Case 2"); break;
default:
Console.WriteLine("Default case"); break;
}
```

Кнопки-переключатели **RadioButton**

При создании программ в Visual Studio для организации разветвлений часто используются компоненты в виде кнопок-переключателей. Состояние такой кнопки (включено - выключено) визуально отражается на форме. Если пользователь выбирает один из вариантов переключателя в группе, все остальные автоматически отключаются.

Группу составляют все элементы управления **RadioButton** в заданном контейнере, таком как **Form**. Чтобы создать на одной форме несколько групп, поместите каждую группу в собственный контейнер, такой как элемент управления **GroupBox** или **Panel**. На форме (рис.3.1) представлены кнопки-переключатели **RadioButton** в контейнере **GroupBox**.

В программу передается номер включенной кнопки (0,1,2,...), который анализируется с помощью оператора **switch**.

Пример написания программы

Задание: ввести три числа - x,y,z. Вычислить

$$U = \begin{cases} y \cdot f(x)^2 + z, & \text{при } z - x = 0 \\ y \cdot e^{f(x)} - z, & \text{при } z - x < 0 \\ y \cdot \sin(f(x)) + z, & \text{при } z - x > 0 \end{cases}$$

В качестве $f(x)$ использовать по выбору: $\sin(x)$, $\cos(x)$, e^x .

Создание формы

Создайте форму, в соответствии с рис. 3.1.

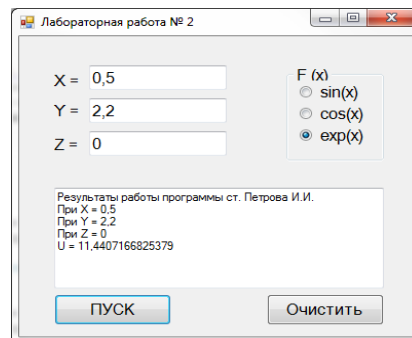


Рис 3.1. Окно лабораторной работы

Выберите в панели элементов из контейнеров **GroupBox** и поместите его в нужное место формы. На форме появится окаймленный линией чистый прямоугольник с заголовком **GroupBox1**. Замените заголовок (**Text**) на **F(x)**. Далее, как показано на рисунке, разместите в данном контейнере три радиокнопки (**RadioButton**). Для первой из них установите свойство **Checked** в значение **True**.

Далее разместите на форме элементы **Label**, **TextBox** и **Button**. Поле для вывода результатов также является элементом **TextBox** с установленным в **True** свойством **Multiline**.

Создание обработчиков событий **FormCreate** и **Button1Click**

Обработчики событий создаются аналогично тому, как и в предыдущих лабораторных работах. Текст обработчика события нажатия на кнопку **ПУСК** приведен ниже.

```
private void button1_Click (object sender, EventArgs e)
{
    // Получение исходных данных из TextBox
    double x = Convert.ToDouble(textBox1.Text);
    double y = Convert.ToDouble(textBox2.Text);
```

```

double z = Convert.ToDouble(textBox3.Text);

// Ввод исходных данных в окно результатов
textBox4.Text = "Результаты работы программы ст. Петрова И.И." +
Environment.NewLine;

textBox4.Text += "При X = " + textBox1.Text + Environment.NewLine;
textBox4.Text += "При Y = " + textBox2.Text + Environment.NewLine;
textBox4.Text += "При Z = " + textBox3.Text + Environment.NewLine;

// Определение номера выбранной функции int n = 0;
if (radioButton2.Checked) n = 1;
else if (radioButton3.Checked) n = 2;

// Вычисление U double u;
switch (n)
{
case 0:
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;
textBox4.Text += "U = " + Convert.ToString(u) + Environment.NewLine;
break;

case 1:
if ((z - x) == 0) u = y * Math.Cos(x) * Math.Cos(x) + z; else if ((z - x) < 0) u =
y*Math.Exp(Math.Cos(x)) - z; else u = y*Math.Sin(Math.Cos(x)) + z;
textBox4.Text += "U = " + Convert.ToString(u) + Environment.NewLine;
break;

case 2:
if ((z - x) == 0) u = y * Math.Exp(x) * Math.Exp(x) + z; else if ((z - x) < 0) u =
y*Math.Exp(Math.Exp(x)) - z; else u = y*Math.Sin(Math.Exp(x)) + z;
textBox4.Text += "U = " + Convert.ToString(u) + Environment.NewLine;
break;

default:

```

```

textBox4.Text += "Решение не найдено" + Environment.NewLine; break;
}
}

```

Запустите программу и убедитесь в том, что все ветви алгоритма выполняются правильно.

Выполнение индивидуального задания

По указанию преподавателя выберите индивидуальное задание из нижеприведенного списка. В качестве $f(x)$ использовать по выбору: $\sin(x)$, x^2 , e^x .

Отредактируйте вид формы и текст программы, в соответствии с полученным заданием.

	$a = \begin{cases} (f(x) + y)^2 - \sqrt{f(x)y}, & xy > 0 \\ (f(x) + y)^2 + \sqrt{ f(x)y }, & xy < 0 \\ (f(x) + y)^2 + 1, & xy = 0. \end{cases}$		$b = \begin{cases} \ln(f(x)) + (f(x)^2 + y)^3, & x/y > 0 \\ \ln f(x)/y + (f(x) + y)^3, & x/y < 0 \\ (f(x)^2 + y)^3, & x = 0 \\ 0, & y = 0. \end{cases}$
	$c = \begin{cases} f(x)^2 + y^2 + \sin(y), & x - y = 0 \\ (f(x) - y)^2 + \cos(y), & x - y > 0 \\ (y - f(x))^2 + \tan(y), & x - y < 0. \end{cases}$		$d = \begin{cases} (f(x) - y)^3 + \arctan(f(x)), & x/y \\ (y - f(x))^3 + \arctan(f(x)), & y/x \\ (y + f(x))^3 + 0.5, & y = x. \end{cases}$
	$e = \begin{cases} i\sqrt{f(x)}, & i - \text{нечетное } x > 0 \\ i/2\sqrt{ f(x) }, & i - \text{четное } x < 0 \\ \sqrt{ if(x) }, & \text{иначе} \end{cases}$		$g = \begin{cases} e^{f(x)- b }, & 0.5 < x < 10 \\ \sqrt{ f(x) + b }, & 0.1 < x < 0.5 \\ 2f(x)^2, & \text{иначе} \end{cases}$
	$s = \begin{cases} e^{f(x)}, & 1 < x < 10 \\ \sqrt{ f(x) + 4*b }, & 12 < x < 40 \\ bf(x)^2, & \text{иначе} \end{cases}$		$j = \begin{cases} \sin(5f(x) + 3\pi f(x)), & -1 < m < x \\ \cos(3f(x) + 5\pi f(x)), & x > m \\ (f(x) + m)^2, & x = m \end{cases}$
	$l = \begin{cases} 2f(x)^3 + 3p^2, & x > p \\ f(x) - p , & 3 < x < p \\ (f(x) - p)^2, & x = p . \end{cases}$		$k = \begin{cases} \ln(f(x) + q), & xq > 10 \\ e^{f(x)+q}, & xq < 10 \\ f(x) + q, & xq = 10 \end{cases}$
	$m = \frac{\max(f(x), y, z)}{\min(f(x), y)} + 5.$		$n = \frac{\min(f(x) + y, y - z)}{\max(f(x), y)}.$

	$p = \frac{ \min(f(x), y) - \max(y, z) }{2}.$		$q = \frac{\max(f(x) + y + z, xyz)}{\min(f(x) + y + z, xyz)}.$
--	---	--	--

ЛАБОРАТОРНАЯ РАБОТА №4. Программирование циклических алгоритмов

Цель лабораторной работы: изучить простейшие средства отладки программ в среде Visual Studio. Составить и отладить программу циклического алгоритма.

Операторы организации циклов

Под циклом понимается многократное выполнение одних и тех же операторов при различных значениях промежуточных данных. Число повторений может быть задано в явной или неявной форме.

К операторам цикла относятся: цикл с предусловием **while**, цикл с постусловием **do while**, цикл с параметром **for** и цикл перебора **foreach**. Рассмотрим некоторые из них.

Цикл с предусловием **while**

Оператор цикла **while** организует выполнение одного оператора (простого или составного) неизвестное заранее число раз. Формат цикла **while**:

while (B) S;

где **B** - выражение, истинность которого проверяется (условие завершения цикла); **S** - тело цикла - оператор (простой или составной).

Перед каждым выполнением тела цикла анализируется значение выражения **B**: если оно истинно, то выполняется тело цикла, и управление передается на повторную проверку условия **B**; если значение **B** ложно - цикл завершается и управление передается на оператор, следующий за оператором **S**.

Если результат выражения **B** окажется ложным при первой проверке, то тело цикла не выполнится ни разу. Отметим, что если условие **B** во время работы цикла не будет изменяться, то возможна ситуация закливания, то есть невозможность выхода из цикла. Поэтому внутри тела должны находиться операторы, приводящие к изменению значения выражения **B** так, чтобы цикл мог корректно завершиться.

В качестве иллюстрации выполнения цикла **while** рассмотрим программу

вывода в консоль целых чисел из интервала от 1 до n.

```
static void Main()
{
    Console.Write("N= ");
    int n=int.Parse(Console.ReadLine()); int i = 1;
    while (i <= n)    //пока i меньше или равно n
        Console.Write(" "+ i++ ); //выводим i на экран, затем увеличиваем его
на 1
}
```

Результаты работы программы:

n ответ

10 1 2 3 4 5 6 7 8 9 10

Цикл с постусловием do while

Оператор цикла **do while** также организует выполнение одного оператора (простого или составного) неизвестное заранее число раз. Однако в отличие от цикла **while** условие завершения цикла проверяется после выполнения тела цикла. Формат цикла **do while**:

```
do S while (B);
```

где **B** - выражение, истинность которого проверяется (условие завершения цикла); **S** - тело цикла - оператор (простой или блок).

Сначала выполняется оператор **S**, а затем анализируется значение выражения **B**: если оно истинно, то управление передается оператору **S**, если ложно - цикл завершается, и управление передается на оператор, следующий за условием **B**. Так как условие **B** проверяется после выполнения тела цикла, то в любом случае тело цикла выполнится хотя бы один раз.

В операторе **do while**, так же, как и в операторе **while**, возможна ситуация закливания в случае, если условие **B** всегда будет оставаться истинным. Цикл с параметром for

Цикл с параметром имеет следующую структуру:

for (<инициализация>; <выражение>; <модификация>)

<оператор>;

Инициализация используется для объявления и/или присвоения начальных значений величинам, используемым в цикле в качестве параметров (счетчиков). В этой части можно записать несколько операторов, разделенных запятой. Областью действия переменных, объявленных в части инициализации цикла, является цикл и вложенные блоки. Инициализация выполняется один раз в начале исполнения цикла.

Выражение определяет условие выполнения цикла: если его результат истинен, цикл выполняется. Истинность выражения проверяется перед каждым выполнением тела цикла, таким образом, цикл с параметром реализован как цикл с предусловием. В блоке выражение через запятую можно записать несколько логических выражений, тогда запятая равносильна операции логическое И (&&).

Модификация выполняется после каждой итерации цикла и служит обычно для изменения параметров цикла. В части модификация можно записать несколько операторов через запятую.

Оператор (простой или составной) представляет собой тело цикла. Любая из частей оператора **for** (инициализация, выражение, модификация, оператор) может отсутствовать, но точку с запятой, определяющую позицию пропускаемой части, надо оставить.

Пример формирования строки, состоящей из чисел от 0 до 9 разделенных пробелами:

```
for (var i = 0; i <= 9; i++)  
{  
s += i + " ";  
}
```

Данный пример работает следующим образом. Сначала вычисляется начальное значение переменной **i**. Затем, пока значение **i** меньше или равно 9,

выполняется тело цикла и затем повторно вычисляется значение i . Когда значение i становится больше 9, условие становится ложным и управление передается за пределы цикла.

Средства отладки программ

Практически в каждой вновь написанной программе после запуска обнаруживаются ошибки.

Ошибки первого уровня (ошибки компиляции) связаны с неправильной записью операторов (орфографические, синтаксические). При обнаружении ошибок компилятор формирует список, который отображается по завершению компиляции (рис. 4.1.). При этом возможен только запуск программы, которая была успешно скомпилирована для предыдущей версии программы.

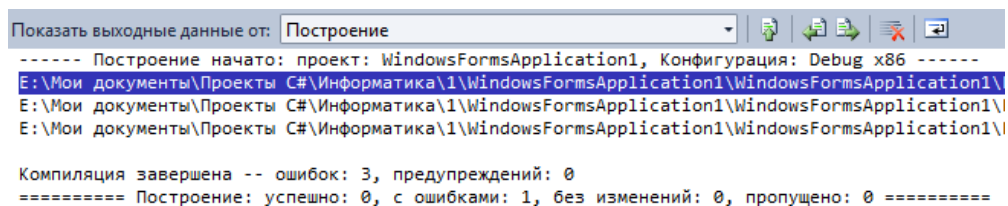


Рис. 4.1. Окно со списком ошибок компиляции.

При выявлении ошибок компиляции в нижней части экрана появляется текстовое окно (рис 4.1.), содержащее сведения обо всех ошибках, найденных в проекте. Каждая строка этого окна содержит имя файла, в котором найдена ошибка, номер строки с ошибкой и характер ошибки. Для быстрого перехода к интересующей ошибке необходимо дважды щелкнуть на строке с ее описанием. Следует обратить внимание на то, что одна ошибка может повлечь за собой другие, которые исчезнут при ее исправлении. Поэтому следует исправлять ошибки последовательно, сверху вниз и, после исправления каждой ошибки компилировать программу снова.

Ошибки второго уровня (ошибки выполнения) связаны с ошибками выбранного алгоритма решения или с неправильной программной реализацией алгоритма. Эти ошибки проявляются в том, что результат расчета оказывается неверным либо происходит переполнение, деление на ноль и др. Поэтому перед

использованием отлаженной программы ее надо протестировать, т.е. сделать просчеты при таких комбинациях исходных данных, для которых заранее известен результат. Если тестовые расчеты указывают на ошибку, то для ее поиска следует использовать встроенные средства отладки среды программирования.

В простейшем случае для локализации места ошибки рекомендуется поступать следующим образом. В окне редактирования текста установить точку останова перед подозрительным участком, которая позволит остановить выполнение программы и далее более детально следить за ходом работы операторов и изменением значений переменных. Для этого достаточно в окне редактирования кода щелкнуть слева от нужной строки. В результате чего данная строка будет выделена красным (рис. 4.2.).

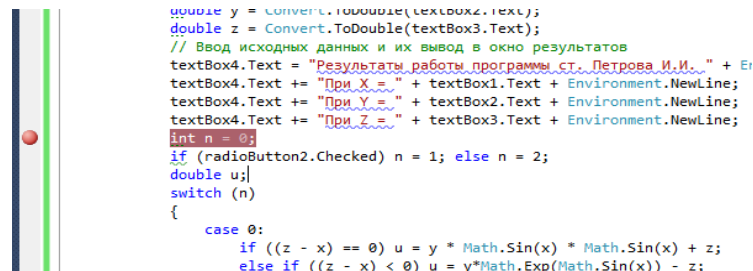


Рис. 4.2. Фрагмент кода с точкой останова

При выполнении программы и достижения установленной точки, программа будет остановлена и далее можно будет выполнять код по операторно с помощью кнопок F10 (без захода в подпрограммы) или F11 (с заходом в подпрограммы) (рис 4.3.).

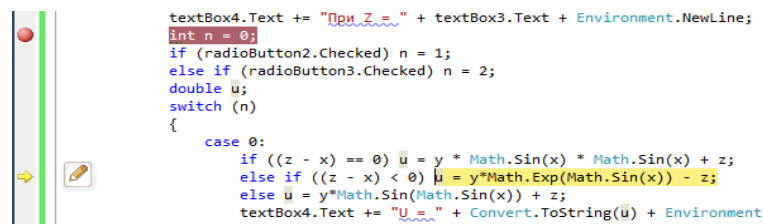


Рис. 4.3. Отладка программы

Желтым выделяется оператор, который будет выполнен. Значение переменных во время выполнения можно увидеть наведя на них курсор. Для снятия программы с выполнения необходимо нажать Shift F5.

Для поиска алгоритмических ошибок контролируются значения

промежуточных переменных на каждом шаге выполнения подозрительного кода и сравниваются с результатами, полученными вручную.

Порядок выполнения задания

Задание: вычислим и выведем на экран таблицу значений функции $y=a \cdot \ln x$ при x , изменяющемся от x_0 до x_k с шагом dx , a - константа.

Панель диалога представлена на рис 4.4.

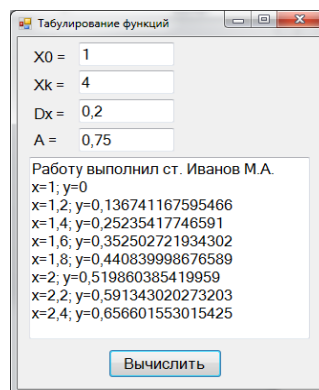


Рис. 4.4. Окно программы для табулирования функции.

Текст обработчика нажатия кнопки **Вычислить** приведен ниже.

```
private void button1_Click (object sender, EventArgs e)
{
    // Считывание начальных данных
    double x0 = Convert.ToDouble(textBox1.Text); double xk =
Convert.ToDouble(textBox2.Text); double dx = Convert.ToDouble(textBox3.Text);
double a = Convert.ToDouble(textBox4.Text);
    textBox5.Text = "Работу выполнил ст. Иванов М.А." +
Environment.NewLine;
    // Цикл для табулирования функции double x = x0;
    while (x<=(xk+dx/2))
    {
        double y = a*Math.Log(x);
        textBox5.Text += "x=" + Convert.ToString(x) +
"; y=" + Convert.ToString(y) + Environment.NewLine; x = x + dx;
    }
}
```

После отладки программы составьте тест ($X_0=2$, $X_k=4$, $Dx=0,2$), установите точку останова на оператор перед циклом и запустите программу в отладочном режиме (F5). После попадания на точку остановки, нажимая клавишу F10, выполните пошагово программу и проследите, как меняются все переменные в процессе выполнения.

ЛАБОРАТОРНАЯ РАБОТА № 5. Программирование с использованием строк

Цель лабораторной работы: изучить правила работы с компонентом `ListBox`. Написать программу для работы со строками.

Тип данных `string`

Для хранения строк в языке C# используется тип `string`. Так, чтобы объявить (и, как правило, сразу инициализировать) строковую переменную, можно написать следующий код:

```
string a = "Текст"; string b = "строки";
```

Над строками можно выполнять операцию сложения – в этом случае текст одной строки будет добавлен к тексту другой:

```
string c = a + " " + b; // Результат: Текст строки
```

Тип `string` на самом деле является псевдонимом для класса `String`, с помощью которого над строками можно выполнять ряд более сложных операций. Например, метод `IndexOf` может осуществлять поиск подстроки в строке, а метод `Substring` возвращает часть строки указанной длины, начиная с указанной позиции:

```
string a = "ABCDEFGHIJKLMNOPQRSTUVWXYZ";
```

```
int index = a.IndexOf("OP"); // Результат: 14 (счёт с 0)
```

```
string b = a.Substring(3, 5); // Результат: DEFGH
```

Если требуется добавить в строку специальные символы, это можно сделать с помощью `escape`-последовательностей, начинающихся с обратного слэша:

Escape-последовательность	Действие
<code>\"</code>	Кавычка
<code>\\</code>	Обратная косая черта
<code>\n</code>	Новая строка
<code>\r</code>	Возврат каретки
<code>\t</code>	Горизонтальная табуляция

Компонент `ListBox`

Компонент **`ListBox`** представляет собой список, элементы которого

выбираются при помощи клавиатуры или мыши. Список элементов задается свойством **Items**. **Items** – это элемент, который имеет свои свойства и свои методы. Методы **Add**, **RemoveAt** и **Insert** используются для добавления, удаления и вставки элементов.

Объект **Items** хранит объекты, находящиеся в списке. Объект может быть любым классом – данные класса преобразуются для отображения в строковое представление методом `ToString`. В нашем случае в качестве объекта будут выступать строки. Однако, поскольку объект **Items** хранит объекты, приведённые к типу `object`, перед использованием необходимо привести их обратно к изначальному типу, в нашем случае `string`:

```
string a = (string)listBox1.Items[0];
```

Для определения номера выделенного элемента используется свойство **SelectedIndex**.

Порядок выполнения индивидуального задания

Задание: Написать программу подсчета числа слов в произвольной строке.

В качестве разделителя может быть любое число пробелов. Для ввода строк использовать **ListBox**. Строки вводятся на этапе проектирования формы, используя окно свойств. Вывод результата организовать в метку **Label**.

Панель диалога будет иметь вид:

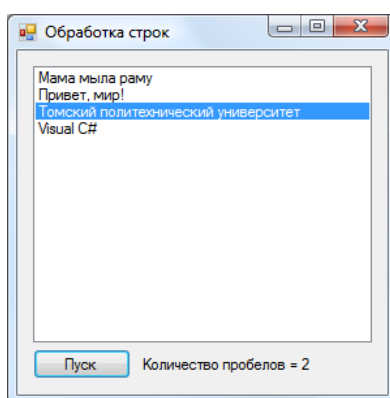


Рис. 5.1. Окно программы обработки строк

Текст обработчика нажатия кнопки «Пуск» приведен ниже.

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

```

// Получаем номер выделенной строки
int index = listBox1.SelectedIndex;
// Считываем строку в переменную str
string str = (string)listBox1.Items[index];
// Узнаем количество символов в строке
int len = str.Length;
// Считаем, что количество пробелов равно 0
int count = 0;
// Устанавливаем счетчик символов в 0
int i = 0;
//Организуем цикл перебора всех символов в строке
while (i < len - 1)
{
// Если нашли пробел, то увеличиваем
// счетчик пробелов на 1
if (str[i] == ' ') count++;
i++;
}
label1.Text = "Количество пробелов = " +
count.ToString();
}

```

Индивидуальные задания

Во всех заданиях исходные данные вводить с помощью **ListBox**. Строки вводятся на этапе проектирования формы, используя окно свойств. Вывод результата организовать в метку **Label**.

Дана строка, состоящая из групп нулей и единиц. Посчитать количество нулей и единиц.

Посчитать в строке количество слов.

Найти количество знаков препинания в исходной строке.

Дана строка символов. Вывести на экран цифры, содержащиеся в строке.

Дана строка символов, состоящая из произвольных десятичных цифр, разделенных пробелами. Вывести количество четных чисел в этой строке.

Дана строка символов. Вывести на экран количество строчных русских букв, входящих в эту строку.

Дана строка символов. Вывести на экран только строчные русские буквы, входящие в эту строку.

Дана строка символов, состоящая из произвольного текста на английском языке, слова разделены пробелами. В каждом слове заменить первую букву на прописную.

Дана строка символов, состоящая из произвольного текста на английском языке, слова разделены пробелами. Удалить первую букву в каждом слове.

Дана строка символов, состоящая из произвольного текста на английском языке, слова разделены пробелами. Поменять местами *i*- и *j*-ю буквы. Для ввода *i* и *j* на форме добавить свои поля ввода.

Дана строка символов, состоящая из произвольного текста на английском языке, слова разделены пробелами. Поменять местами первую и последнюю буквы каждого слова.

Дана строка символов, состоящая из произвольного текста на английском языке, слова разделены пробелами. Заменить все буквы латинского алфавита на знак $\equiv$.

Дана строка символов, содержащая некоторый текст на русском языке. Заменить все большие буквы $\equiv$ на символ $\equiv$.

Дана строка символов, содержащая некоторый текст. Разработать программу, которая определяет, является ли данный текст палиндромом, т.е. читается ли он слева направо так же, как и справа налево (например, «А роза упала на лапу Азора»). Дана строка символов, состоящая из произвольного текста на английском языке, слова разделены пробелами. Сформировать новую строку, состоящую из чисел длин слов в исходной строке.

ЛАБОРАТОРНАЯ РАБОТА № 6. Программирование с использованием одномерных массивов

Цель лабораторной работы: изучить способы получения случайных чисел. Написать программу для работы с одномерными массивами.

Работа с массивами

Массив - набор элементов одного и того же типа, объединенных общим именем. Массивы в C# можно использовать по аналогии с тем, как они используются в других языках программирования. Однако C#-массивы имеют существенные отличия: они относятся к ссылочным типам данных, более того - реализованы как объекты. Фактически имя массива является ссылкой на область кучи (динамической памяти), в которой последовательно размещается набор элементов определенного типа. Выделение памяти под элементы происходит на этапе инициализации массива. А за освобождением памяти следит система сборки мусора - неиспользуемые массивы автоматически утилизируются данной системой.

Рассмотрим в данной лабораторной работе одномерные массивы. *Одномерный массив* - это фиксированное количество элементов одного и того же типа, объединенных общим именем, где каждый элемент имеет свой номер. Нумерация элементов массива в C# начинается с нуля, то есть, если массив состоит из 10 элементов, то его элементы будут иметь следующие номера: 0, 1, 2, 3, 4, 5, 6, 7, 8, 9.

Одномерный массив в C# реализуется как объект, поэтому его создание представляет собой двухступенчатый процесс. Сначала объявляется ссылочная переменная на массив, затем выделяется память под требуемое количество элементов базового типа, и ссылочной переменной присваивается адрес нулевого элемента в массиве. Базовый тип определяет тип данных каждого элемента массива. Количество элементов, которые будут храниться в массиве, определяется размер массива. В общем случае процесс объявления переменной типа массив, и выделение необходимого объема памяти может быть разделено.

Кроме того, на этапе объявления массива можно произвести его инициализацию. Поэтому для объявления одномерного массива может использоваться одна из следующих форм записи:

базовый_тип [] имя_массива;

Описана ссылка на одномерный массив, которая в дальнейшем может быть использована для адресации на уже существующий массив. Например:

int [] a;

базовый_тип [] имя_массива = new базовый_тип [размер];

Объявлен одномерный массив заданного типа и выделена память под одномерный массив указанной размерности. Адрес данной области памяти записан в ссылочную переменную. Элементы массива равны нулю (В C# элементам массива присваиваются начальные значения по умолчанию в зависимости от базового типа. Для арифметических типов - нули, для ссылочных типов - null, для символов - пробел). Например: **int []a=new int [10];**

базовый_тип [] имя_массива = {список инициализации};

Выделена память под одномерный массив, размерность которого соответствует количеству элементов в списке инициализации. Адрес этой области памяти записан в ссылочную переменную. Значение элементов массива соответствует списку инициализации. Например: **int []a={0, 1, 2, 3};**

Обращения к элементам массива происходит с помощью индекса, для этого нужно указать имя массива и в квадратных скобках его номер. Например, **a[0], b[10], c[i]**.

Так как массив представляет собой набор элементов, объединенных общим именем, то обработка массива обычно производится в цикле. Например:

int[] myArray = { 0, 1, 2, 3, 4, 5, 6, 7, 8, 9 };

int i; for (i = 0; i < 10; ++i)

Console.WriteLine(myArray[i]);

Случайные числа

Одним из способов задания массива является задание определение

элементов через случайные числа. Для работы со случайными числами используются в основном два метода класса **Random: Random и Next**. Метод **Random** подготавливает работу со случайными числами, обеспечивая, надежный способ создания непредсказуемой последовательности чисел.

Метод **Random.Next** создает случайное число в диапазоне значений от нуля до **Int32.MaxValue**. Для создания случайного числа в диапазоне от нуля до какого-либо другого положительного числа используется перегрузка метода **Random.Next(Int32)**. Для создания случайного числа в другом диапазоне используется перегрузка метода **Random.Next(Int32, Int32)**.

Порядок выполнения индивидуального задания

Создайте форму с элементами управления как приведено на рис. 6.1. Опишите одномерный массив. Создайте обработчики события для кнопок (код приведен ниже). Данная программа заменяет все отрицательные числа нулями. Протестируйте правильность выполнения программы. Модифицируйте программу в соответствии с индивидуальным заданием.

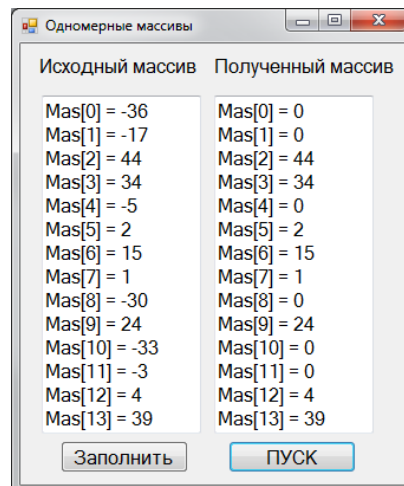


Рис. 6.1. Окно программы для работы с одномерными массивами

`using System;`

`using System.Collections.Generic; using System.ComponentModel; using`

`System.Data;`

`using System.Drawing; using System.Linq; using System.Text;`

`using System.Windows.Forms;`

```

namespace WindowsFormsApplication1
{
    public partial class Form1 : Form
    {
        int[] Mas = new int[15];
        public Form1()
        {
            InitializeComponent();
        }

        private void button1_Click(object sender, EventArgs e)
        {
            Random rand = new Random();
            textBox1.Text = "";
            for (int i = 0; i < 15; i++)
            {
                Mas[i] = rand.Next(-50, 50);
                textBox1.Text += "Mas[" + Convert.ToString(i) + "] = "
                + Convert.ToString(Mas[i]) + Environment.NewLine;
            }
        }

        private void button2_Click(object sender, EventArgs e)
        {
            textBox2.Text = "";
            for (int i = 0; i < 15; i++)
            {
                Environment.NewLine;
            }
            if (Mas[i] < 0) Mas[i] = 0;
            textBox2.Text += "Mas[" + Convert.ToString(i) + "] = "
            + Convert.ToString(Mas[i]) + }
        }
    }
}

```

6.3. Варианты заданий

В массиве из 20 целых чисел найти наибольший элемент и поменять его местами с первым элементом.

В массиве из 10 целых чисел найти наименьший элемент и поменять его местами с последним элементом.

В массиве из 15 вещественных чисел найти наибольший элемент и поменять его местами с последним элементом.

В массиве из 25 вещественных чисел найти наименьший элемент и поменять его местами с первым элементом.

Дан массив X , содержащий 27 элементов. Вычислить и вывести элементы нового массива Y , где $y_i = 6.85x_i^2 - 1.52$. Если $y_i < 0$, то вычислить и вывести $a = x_i^3 - 0.62$ и продолжить вычисления; если $y_i \in 0$, то вычислить и вывести $b = 1/x_i^2$ и продолжить вычисления.

Дан массив X , содержащий 16 элементов. Вычислить и вывести

$$d_i = \frac{e^{x_i} + 2e^{-x_i}}{\sqrt{5 + \sin x_i}}$$

значения d_i , где d_i и значения $d_i > 0.1$.

Дан массив Y , содержащий 25 элементов. Записать в массив R и

$$r_i = \frac{5y_i + \cos^2 y_i}{2.35}$$

вывести значения элементов, вычисляемые по формуле r_i , $i=1,2,\dots,25$.

Дан массив F , содержащий 18 элементов. Вычислить и вывести элементы нового массива $p_i = 0.13f_i^3 - 2.5f_i + 8$. Вывести отрицательные элементы массива P .

Вычислить и вывести элементы массива Z , где $z_i = i^2 + 1$, если i – нечетное, и $z_i = 2i - 1$, если i – четное. Сформировать и вывести массив D : $d_i = 2.5z_i$, если $z_i < 2.5$ и $d_i = z_i/2.5$, если $z_i \in 2.5$.

Заданы массивы D и E . Вычислить и вывести значения $f_i = (2d_i + \sin e_i)/d_i$, где $i=1,2,\dots,16$; вывести $1 < f_i < 3$. В массиве R , содержащем 25 элементов, заменить значения отрицательных элементов квадратами значений, значения

положительных увеличить на 7, а нулевые значения оставить без изменения. Вывести массив R.

Дан массив A целых чисел, содержащий 30 элементов. Вычислить и вывести сумму тех элементов, которые кратны 5.

Дан массив A целых чисел, содержащий 30 элементов. Вычислить и вывести сумму тех элементов, которые нечетны и отрицательны.

Дан массив A целых чисел, содержащий 30 элементов. Вычислить и вывести сумму тех элементов, которые удовлетворяют условию $|a_i| < i^2$.

Дан массив A целых чисел, содержащий 30 элементов. Вычислить и вывести количество и сумму тех элементов, которые делятся на 5 и не делятся на 7.

Дан массив A вещественных чисел, содержащий 25 элементов. Вычислить и вывести число отрицательных элементов и число членов, принадлежащих отрезку [1,2].

Дан массив C, содержащий 23 элемента. Вычислить и вывести среднее арифметическое всех значений $c_i > 3.5$.

Дан массив Z целых чисел, содержащий 35 элементов. Вычислить и вывести $R=S+P$, где S – сумма четных элементов, меньших 3, P – произведение нечетных элементов, больших 1.

Дан массив Q натуральных чисел, содержащий 20 элементов. Найти и вывести те элементы, которые при делении на 7 дают остаток 1,2 или 5.

Дан массив Q натуральных чисел, содержащий 20 элементов. Найти и вывести те элементы, которые обладают тем свойством, что корни уравнения $q_i^2 + 3q_i - 5 = 0$ действительны и положительны.

Дан массив, содержащий 10 элементов. Вычислить произведение элементов, стоящих после первого отрицательного элемента. Вывести исходный массив и результат вычислений. Дан массив, содержащий 14 элементов. Вычислить сумму элементов, стоящих до первого отрицательного элемента. Вывести исходный массив и результат вычислений.

Дан массив, содержащий 12 элементов. Все четные элементы сложить, вывести массив и результат.

Дан массив, содержащий 15 элементов. Все положительные элементы возвести в квадрат, а отрицательные умножить на 2. Вывести исходный и полученный массив.

Дан массив, содержащий 14 элементов. Все отрицательные элементы заменить на 3. Вывести исходный и полученный массив.

ЛАБОРАТОРНАЯ РАБОТА № 7. Программирование с использованием многомерных массивов

Цель лабораторной работы: изучить свойства компонента **dataGridView**. Написать программу с использованием двумерных массивов.

Двухмерные массивы

Многомерные массивы имеют более одного измерения. Чаще всего используются двумерные массивы, которые представляют собой таблицы. Каждый элемент массива имеет два индекса, первый определяет номер строки, второй - номер столбца, на пересечении которых находится элемент. Нумерация строк и столбцов начинается с нуля. Объявить двумерный массив можно одним из предложенных способов:

тип [,] имя_массива;

тип [,] имя_массива = new тип [размер1, размер2];

тип [,] имя_массива = {{элементы 1-ой строки}, ..., {элементы n-ой строки}};
тип [,] имя_массива = new тип [,] {{элементы 1-ой строки}, ..., {элементы n-ой строки}};
строки}};
строки}};

Пример кода, использующего многомерные массивы:

```
// объявление и инициализация двумерного массива
int[,] array2D = new int[,] { { 1, 2 }, { 3, 4 }, { 5, 6 }, { 7, 8 } };
// Объявление такого массива с указанием размерности (кол-во строки столбцов)
int[,] array2Da = new int[4, 2] { { 1, 2 }, { 3, 4 }, { 5, 6 }, { 7, 8 } };
// Объявление двумерного массива элементами, которого являются строки
string[,] array2Db = new string[3, 2] { { "one", "two" }, { "three", "four" }, { "five", "six" } };
// Объявление трехмерного массива
int[ , , ] array3D = new int[ , , ] { { { 1, 2, 3 }, { 4, 5, 6 } }, { { 7, 8, 9 }, { 10, 11, 12 } } };
// Объявление трехмерного массива с указанием размерности
int[ , , ] array3Da = new int[2, 2, 3] { { { 1, 2, 3 }, { 4, 5, 6 } }, { { 7, 8, 9 }, { 10, 11, 12 } } };
// Доступ к элементам массива
```

```

System.Console.WriteLine(array2D[0, 0]);
System.Console.WriteLine(array2D[0, 1]);
System.Console.WriteLine(array2D[1, 0]);
System.Console.WriteLine(array2D[1, 1]);
System.Console.WriteLine(array2D[3, 0]);
System.Console.WriteLine(array2Db[1, 0]);
System.Console.WriteLine(array3Da[1, 0, 1]);
System.Console.WriteLine(array3D[1, 1, 2]);
// Результаты работы программы (выводятся в консоль):
// 1
// 2
// 3
// 4
// 7
// three
// 8
// 12

```

Элемент управления DataGridView

При работе с двумерными массивами ввод и вывод информации на экран удобно организовывать в виде таблиц. Элемент управления **DataGridView** может быть использован для отображения информации в виде двумерной таблицы. Для обращения к ячейке в этом элементе необходимо указать номер строки и номер столбца. Например: **dataGridView1.Rows[2].Cells[7].Value = "*" ;** данный код позволяет записать во вторую строку в 7 ячейку знак звездочка.

Порядок выполнения задания

Задание: создать программу для определения целочисленной матрицы

15 на 15. Разработать обработчик для поиска минимального элемента на дополнительной диагонали матрицы. Результат, после нажатия кнопки типа **Button**, вывести в **textBox**.

Окно программы приведено на рис. 7.1.

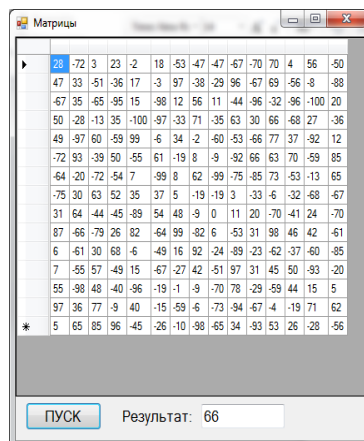


Рис. 7.1. Окно программы для работы с двумерным массивом
Текст обработчика события нажатия на кнопку приведен ниже.

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

столбцов

```
dataGridView1.RowCount = 15; //Указываем количество строк
dataGridView1.ColumnCount = 15; //Указываем количество
```

```
int[,] a = new int[15,15]; //Инициализируем массив
int i,j;
//Заполняем матрицу случайными числами
Random rand = new Random(); for (i=0; i<15; i++)
for (j=0; j<15; j++)
a[i,j] = rand.Next(-100,100);
//Выводим матрицу в dataGridView1 for (i=0; i<15; i++)
for (j=0; j<15; j++) dataGridView1.Rows[i].Cells[j].Value
=Convert.ToString(a[i,j]);
//производим поиск максимального элемента на
```

дополнительной диагонали

```
int m = int.MinValue; for (i = 0; i < 15; i++)  
if (a[i, 14 - i] > m) m = a[i, 14 - i];  
// выводим результат  
textBox1.Text = Convert.ToString(m);  
}
```

Индивидуальные задания

Дана матрица A (3,4). Найти наименьший элемент в каждой строке матрицы. Вывести исходную матрицу и результаты вычислений.

Дана матрица A (3,3). Вычислить сумму второй строки и произведение первого столбца. Вывести исходную матрицу и результаты вычислений.

Дана матрица A (4,4). Найти наибольший элемент в главной диагонали. Вывести матрицу и наибольший элемент.

Дана матрица A (3,4). Найти сумму элементов главной диагонали и эту сумму поставить на место последнего элемента. Вывести исходную и полученную матрицу.

Дана матрица A (4,3). Вычислить наибольший элемент матрицы. Вывести исходную матрицу и наибольший элемент.

Дана матрица A (4,3). Найти количество положительных элементов.

Дана матрица A (3,4). Найти количество отрицательных элементов.

Даны матрицы X (15,15) и Y (15,15). Вычислить и вывести элементы новой матрицы $z_{ij}=12x_{ij}-0.85y_{ij}^2$.

Даны матрицы A (6,6), B (6,6) и C (6,6). Получить матрицу D (6,6), элементы которой вычисляются по формуле $d_{ij}=\max\{a_{ij},(b_{ij}+c_{ij})\}$. Матрицу D (6,6) вывести.

Вычислить сумму S элементов главной диагонали матрицы B (10,10). Если $S>10$, то исходную матрицу преобразовать по формуле $b_{ij}=b_{ij}+13.5$; если $S\leq 10$, то $b_{ij}=b_{ij}^2-1.5$. Вывести сумму S и преобразованную матрицу.

Дана матрица F (15,15). Вывести номер и среднее арифметическое

элементов строки, начинающейся с 1. Если такой строки нет, то вывести сообщение —строки нет

Дана матрица $F(7,7)$. Найти наименьший элемент в каждом столбце. Вывести матрицу и найденные элементы.

Найти наибольший элемент главной диагонали матрицы $A(15,15)$ и вывести всю строку, в которой он находится.

Найти наибольшие элементы каждой строки матрицы $Z(16,16)$ и поместить их на главную диагональ. Вывести полученную матрицу.

Вычислить суммы элементов матрицы $Y(12,12)$ по столбцам и вывести их.

Найти наибольший элемент матрицы $A(10,10)$ и записать нули в ту строку и столбец, где он находится. Вывести наибольший элемент, исходную и полученную матрицу.

Дана матрица $R(9,9)$. Найти наименьший элемент в каждой строке и записать его на место первого элемента строки. Вывести исходную и полученную матрицы.

Определить количество положительных элементов каждой строки матрицы $A(10,20)$ и запомнить их в одномерном массиве N . Массив N вывести.

Вычислить количество N положительных элементов последнего столбца матрицы $X(5,5)$. Если $N < 3$, то вывести все положительные элементы матрицы, если $N \geq 3$, то вывести сумму элементов главной диагонали матрицы.

Вычислить и вывести сумму элементов матрицы $A(12,12)$, расположенных над главной диагональю матрицы.

ЛАБОРАТОРНАЯ РАБОТА № 8. Программирование с использованием средств для отображения графической информации

Цель лабораторной работы: изучить возможности построения графиков с помощью компонента отображения графической информации **Chart**. Написать и отладить программу построения на экране графика заданной функции.

Как строится график с помощью компонента Chart

Обычно результаты расчетов представляются в виде графиков и диаграмм. Библиотека .NET Framework имеет мощный элемент управления Chart для отображения на экране графической информации (рис. 8.1).

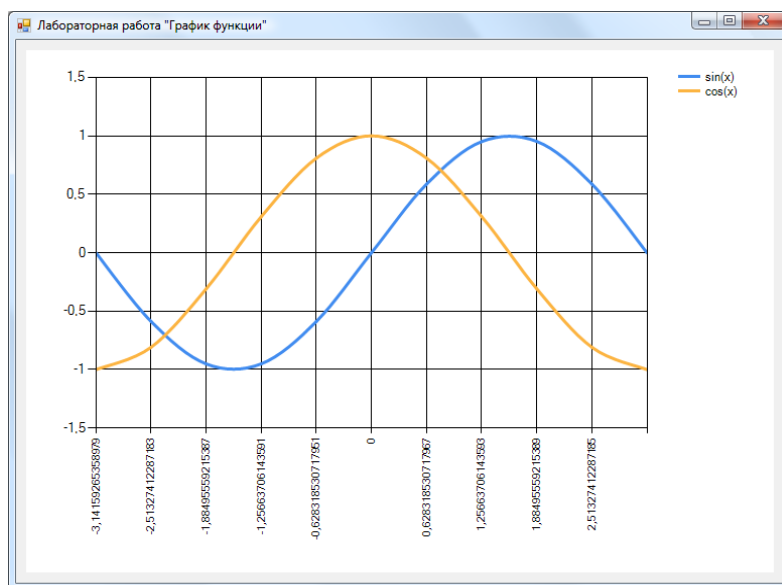


Рис 8.1. Окно программы с элементом управления.

Построение графика (диаграммы) производится после вычисления таблицы значений функции $y=f(x)$ на интервале $[X_{min}, X_{max}]$ с заданным шагом. Полученная таблица передается в специальный массив Points объекта Series компонента Chart с помощью метода DataBindXY. Компонент Chart осуществляет всю работу по отображению графиков: строит и размечает оси, рисует координатную сетку, подписывает название осей и самого графика, отображает переданную таблицу в виде всевозможных графиков или диаграмм. При необходимости компоненту Chart передаются данные о толщине, стиле и цвете линий, параметрах шрифта подписей, шагах разметки координатной сетки и другие настройки. В процессе работы программы изменение параметров

возможно через обращение к соответствующим свойствам компонента Chart. Так, например, свойство AxisX содержит значение максимального предела нижней оси графика и при его изменении во время работы программы автоматически изменяется изображение графика.

Пример написания программы

Задание: составить программу, отображающую графики функций $\sin(x)$ и $\cos(x)$ на интервале $[X_{\min}, X_{\max}]$. Предусмотреть возможность изменения разметки координатных осей, а также шага построения таблицы.

Прежде всего, следует определить в коде класса все необходимые переменные и константы. Конечно, можно обойтись и без этого, вставляя значения в виде чисел прямо в формулы, но это, во-первых, снизит читабельность кода программы, а во вторых, значительно усложнит изменение каких-либо параметров программы, например, интервала построения графика.

```
/// <summary>
/// Левая граница графика
/// </summary>
private double XMin = -Math.PI;
```

```
/// <summary>
/// Правая граница графика
/// </summary>
private double XMax = Math.PI;
```

```
/// <summary>
/// Шаг графика
```

```
/// </summary>
```

```
private double Step = (Math.PI * 2) / 10;
// Массив значений X - общий для обоих графиков
private double[] x;
// Два массива Y - по одному для каждого графика
private double[] y1; private double[] y2;
```

Также в коде класса следует описать глобальную переменную типа Chart, к которой мы будем обращаться из разных методов:

Chart chart;

Поскольку данный класс не входит в пространства имен, подключаемые по умолчанию, следует выполнить дополнительные действия. Во-первых, в Обозревателе решений нужно щёлкнуть правой кнопкой по секции Ссылки и добавить ссылку на библиотеку визуализации (рис. 8.2):

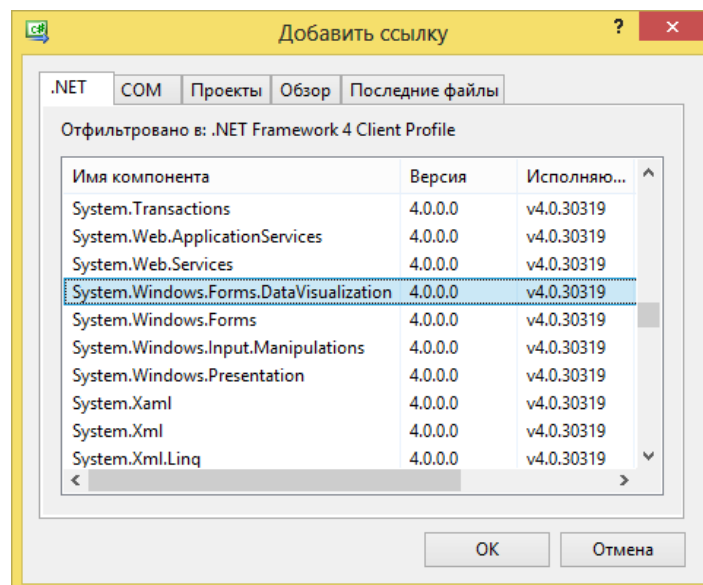


Рис. 8.2. Добавление ссылки на библиотеку визуализации.

Кроме того, следует подключить соответствующее пространство имен:

using

System.Windows.Forms.DataVisualization.Charting;

Далее следует определить метод, который будет рассчитывать количество шагов и вычислять значения функций в каждой точке, внося вычисленные значения в массивы x, y1 и y2:

```
/// <summary>
/// Расчёт значений графика
/// </summary>
```

```

private void CalcFunction()
{
    // Количество точек графика
    int count = (int)Math.Ceiling((XMax - XMin) / Step)
    + 1;
    // Создаём массивы нужных размеров
    x = new double[count]; y1 = new double[count]; y2 =
new double[count];
    // Расчитываем точки для графиков функции
    for (int i = 0; i < count; i++)
    {
        // Вычисляем значение X
        x[i] = XMin + Step * i;
        // Вычисляем значение функций в точке X
        y1[i] = Math.Sin(x[i]);
        y2[i] = Math.Cos(x[i]);
    }
}

```

После расчёта значений нужно отобразить графики на форме с помощью элемента Chart. Элемент управления Chart нельзя выбрать с помощью панели элементов – его нужно создавать прямо в коде программы. Вторым шагом следует создать область отображения графика и настроить внешний вид осей:

```

/// <summary>
/// Создаём элемент управления Chart и настраиваем его
/// </summary>
private void CreateChart()
{
    // Создаём новый элемент управления Chart
    chart = new Chart();
    // Помещаем его на форму

```

```
chart.Parent = this;
// Задаём размеры элемента
chart.SetBounds(10, 10, ClientSize.Width - 20,
ClientSize.Height - 20);

// Создаём новую область для построения графика
ChartArea area = new ChartArea();
// Даём ей имя (чтобы потом добавлять графики)
area.Name = "myGraph";
// Задаём левую и правую границы оси X area.AxisX.Minimum = XMin;
area.AxisX.Maximum = XMax;
// Определяем шаг сетки
area.AxisX.MajorGrid.Interval = Step;
// Добавляем область в диаграмму
chart.ChartAreas.Add(area);

// Создаём объект для первого графика
Series series1 = new Series();
// Ссылаемся на область для построения графика
series1.ChartArea = "myGraph";
// Задаём тип графика - сплайны
series1.ChartType = SeriesChartType.Spline;
// Указываем ширину линии графика
series1.BorderWidth = 3;
// Название графика для отображения в легенде
series1.LegendText = "sin(x)";
// Добавляем в список графиков диаграммы
```

```
chart.Series.Add(series1);
```

```
// Аналогичные действия для второго графика Series series2 = new Series();  
series2.ChartArea = "myGraph"; series2.ChartType = SeriesChartType.Spline;  
series2.BorderWidth = 3;
```

```
series2.LegendText = "cos(x)"; chart.Series.Add(series2);
```

```
// Создаём легенду, которая будет показывать названия  
Legend legend = new Legend (); chart.Legends.Add(legend);  
}
```

Наконец, все эти методы следует откуда-то вызвать. Чтобы графики появлялись сразу после запуска программы, надо вызывать их в обработчике события Load формы:

```
private void Form1_Load (object sender, EventArgs e)  
{  
    // Создаём элемент управления  
    CreateChart();  
  
    // Расчитываем значения точек графиков функций  
    CalcFunction();  
  
    // Добавляем вычисленные значения в графики  
    chart.Series[0].Points.DataBindXY(x, y1);  
    chart.Series[1].Points.DataBindXY(x, y2);  
}
```

ЛАБОРАТОРНАЯ РАБОТА № 9. Программирование графики

Цель лабораторной работы: изучить возможности Visual Studio по созданию простейших графических изображений. Написать и отладить программу построения на экране различных графических примитивов.

Сообщение WM_PAINT

Прежде чем приступить к описанию способов рисования в окнах, применяемых приложениями .NET Frameworks, расскажем о том, как это делают «классические» приложения Microsoft Windows.

ОС Microsoft Windows следит за перемещением и изменением размера окон и при необходимости извещает приложения, о том, что им следует перерисовать содержимое окна. Для извещения в очередь приложения записывается сообщение с идентификатором **WM_PAINT**. Получив такое сообщение, функция окна должна выполнить перерисовку всего окна или его части, в зависимости от дополнительных данных, полученных вместе с сообщением **WM_PAINT**.

Для облегчения работы по отображению содержимого окна весь вывод в окно обычно выполняют в одном месте приложения — при обработке сообщения **WM_PAINT** в функции окна. Приложение должно быть сделано таким образом, чтобы в любой момент времени при поступлении сообщения **WM_PAINT** функция окна могла перерисовать все окно или любую его часть, заданную своими координатами.

Последнее нетрудно сделать, если приложение будет хранить где-нибудь в памяти свое текущее состояние, пользуясь которым функция окна сможет перерисовать окно в любой момент времени.

Здесь не имеется в виду, что приложение должно хранить образ окна в виде графического изображения и восстанавливать его при необходимости, хотя это и можно сделать. Приложение должно хранить информацию, на основании которой оно может в любой момент времени перерисовать окно. Сообщение **WM_PAINT** передается функции окна, если стала видна область окна, скрытая

раньше другими окнами, если пользователь изменил размер окна или выполнил операцию прокрутки изображения в окне. Приложение может передать функции окна сообщение **WM_PAINT** явным образом, вызывая функции программного интерфейса Win32 API, такие как **UpdateWindow**, **InvalidateRect** или **InvalidateRgn**.

Иногда ОС Microsoft Windows может сама восстановить содержимое окна, не посылая сообщение **WM_PAINT**. Например, при перемещении курсора мыши или значка свернутого приложения ОС восстанавливает содержимое окна. Если же ОС не может восстановить окно, функция окна получает от ОС сообщение **WM_PAINT** и перерисовывает окно самостоятельно.

Событие Paint

Для форм класса **System.Windows.Forms** предусмотрен удобный объектно-ориентированный способ, позволяющий приложению при необходимости перерисовывать окно формы в любой момент времени. Когда вся клиентская область окна формы или часть этой области требует перерисовки, форме передается событие **Paint**. Все, что требуется от программиста, это создать обработчик данного события (рис. 8.1.), наполнив его необходимой функциональностью.

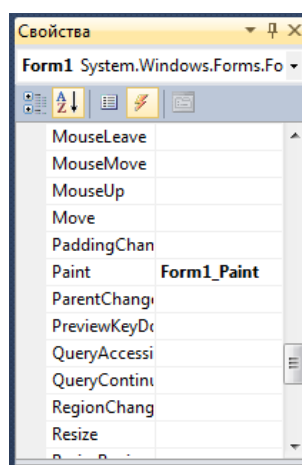


Рис. 9.1. Создание обработчика события Paint

Объект Graphics для рисования

Перед тем как рисовать линии и фигуры, отображать текст, выводить изображения и управлять ими в GDI необходимо создать объект **Graphics**.

Объект **Graphics** представляет поверхность рисования GDI и используется для создания графических изображений. Ниже представлены два этапа работы с графикой.

Создание объекта **Graphics**.

Использование объекта **Graphics** для рисования линий и фигур, отображения текста или изображения и управления ими.

Существует несколько способов создания объектов **Graphics**. Одним из самых используемых является получение ссылки на объект **Graphics** через объект **PaintEventArgs** при обработке события **Paint** формы или элемента управления:

```
private void Form1_Paint(object sender, PaintEventArgs e)
{
    Graphics g = e.Graphics; // Объявляется объект Graphics
    // Далее вставляется код рисования
}
```

Методы и свойства класса **Graphics**

Имена большого количества методов, определенных в классе **Graphics**, начинаются с префикса **Draw*** и **Fill***. Первые из них предназначены для рисования текста, линий и не закрашенных фигур (таких, например, как прямоугольные рамки), а вторые — для рисования закрашенных геометрических фигур. Мы рассмотрим применение только самых важных из этих методов, а полную информацию Вы найдете в документации.

Метод **DrawLine** рисует линию, соединяющую две точки с заданными координатами. Ниже мы привели прототипы различных перегруженных версий этого метода:

```
public void DrawLine(Pen, Point, Point); public void DrawLine(Pen, PointF, PointF);
```

```
public void DrawLine(Pen, int, int, int, int);
```

```
public void DrawLine(Pen, float, float, float, float);
```

Первый параметр задает инструмент для рисования линии — перо.

Перья создаются как объекты класса **Pen**, например:

Pen p = new Pen (Brushes.Black,2);

Здесь мы создали черное перо толщиной 2 пиксела. Создавая перо, Вы можете выбрать его цвет, толщину и тип линии, а также другие атрибуты.

Остальные параметры перегруженных методов **DrawLine** задают координаты соединяемых точек. Эти координаты могут быть заданы как объекты класса **Point** и **PointF**, а также в виде целых чисел и чисел с плавающей десятичной точкой.

В классах **Point** и **PointF** определены свойства **X** и **Y**, задающие, соответственно, координаты точки по горизонтальной и вертикальной оси. При этом в классе **Point** эти свойства имеют целочисленные значения, а в классе **PointF** — значения с плавающей десятичной точкой.

Третий и четвертый вариант метода **DrawLine** позволяет задавать координаты соединяемых точек в виде двух пар чисел. Первая пара определяет координаты первой точки по горизонтальной и вертикальной оси, а вторая — координаты второй точки по этим же осям. Разница между третьим и четвертым методом заключается в использовании координат различных типов (целочисленных **int** и с плавающей десятичной точкой **float**).

Чтобы испытать метод **DrawLine** в работе, создайте приложение **DrawLineApp** (аналогично тому, как Вы создавали предыдущее приложение). В этом приложении создайте следующий обработчик события **Paint**:

```
private void Form1_Paint(object sender, PaintEventArgs e)
{
    Graphics g = e.Graphics; g.Clear(Color.White);
    for (int i = 0; i < 50; i++)
    {
        20);
        g.DrawLine(new Pen(Brushes.Black, 2), 10, 4 * i + 20, 200, 4 * i +

    }
}
```

Здесь мы вызываем метод **DrawLine** в цикле, рисуя 50 горизонтальных линий (рис. 9.2.).

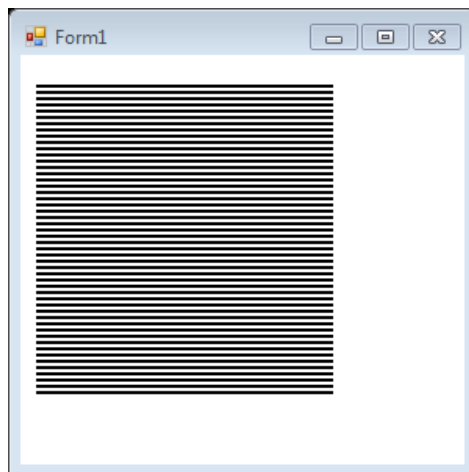


Рис. 9.2. Пример использования DrawLine

Вызвав один раз метод **DrawLines**, можно нарисовать сразу несколько прямых линий, соединенных между собой. Иными словами, метод **DrawLines** позволяет соединить между собой несколько точек. Координаты этих точек по горизонтальной и вертикальной оси передаются методу через массив класса **Point** или **PointF**:

public void DrawLines(Pen, Point[]); public void DrawLines(Pen, PointF[];

Для демонстрации возможностей метода **DrawLines** создайте приложение. Создайте кисть **pen** для рисования линий:

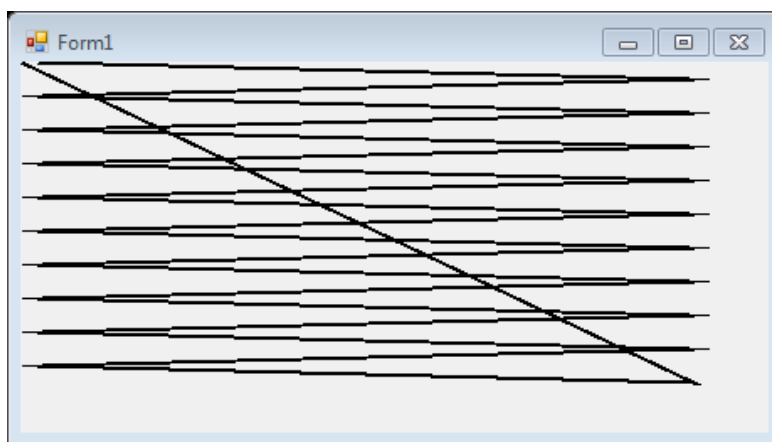


Рис. 9.3. Пример использования массива точек

Для прорисовки прямоугольников можно использовать метод **DrawRectangle(Pen, int, int, int, int);** В качестве первого параметра передается перо класса **Pen**. Остальные параметры задают расположение и размеры прямоугольника.

Для прорисовки многоугольников можно использовать метод

DrawPolygon (Pen, Point[]);

Метод **DrawEllipse** рисует эллипс, вписанный в прямоугольную область, расположение и размеры которой передаются ему в качестве параметров. При помощи метода **DrawArc** программа может нарисовать сегмент эллипса. Сегмент задается при помощи координат прямоугольной области, в которую вписан эллипс, а также двух углов, отсчитываемых в направлении против часовой стрелки. Первый угол **Angle1** задает расположение одного конца сегмента, а второй **Angle2** — расположение другого конца сегмента

В классе **Graphics** определен ряд методов, предназначенных для рисования закрашенных фигур. Имена некоторых из этих методов, имеющих префикс **Fill**:

FillRectangle (рисование закрашенного прямоугольника), **FillRectangles** (рисование множества закрашенных многоугольников), **FillPolygon** (рисование закрашенного многоугольника), **FillEllipse** (рисование закрашенного эллипса) **FillPie** (рисование закрашенного сегмента эллипса) **FillClosedCurve** (рисование закрашенного сплайна) **FillRegion** (рисование закрашенной области типа **Region**).

Есть два отличия методов с префиксом **Fill** от одноименных методов с префиксом **Draw**. Прежде всего, методы с префиксом **Fill** рисуют закрашенные фигуры, а методы с префиксом **Draw** — не закрашенные. Кроме этого, в качестве первого параметра методам с префиксом **Fill** передается не перо класса **Pen**, а кисть класса **Brush**.

Как видите платформа .Net содержит большое число классов со многими методами и свойствами. Нет смысла описывать все классы, методы в каком-либо учебнике или в данном пособии, поскольку по любому методу или классу можно получить MSDN справку набрав наименование метода в среде Visual Studio и нажав на нем клавишу F1. Также, при наборе метода в редакторе кода сред показывает краткую справку о передаваемых параметрах.

Выполнение индивидуального задания

Изучите с помощью справки MSDN методы и свойства классов **Graphics**,

Color, Pen и SolidBrush. Создайте собственное приложение выводящий на форму рисунок, состоящий из различных объектов (линий, многоугольников, эллипсов, прямоугольников и пр.), не закрашенных и закрашенных полностью. Используйте разные цвета и стили линий (сплошные, штриховые, штрих - пунктирные).

ЛАБОРАТОРНАЯ РАБОТА № 10. Простейшая анимация

Цель лабораторной работы: изучить возможности Visual Studio по созданию простейшей анимации. Написать и отладить программу, выводящую на экран анимационное изображение.

Работа с таймером

Класс для работы с таймером (Timer) формирует в приложении повторяющиеся события. События повторяются с периодичностью, указанной в миллисекундах, в свойстве **Interval**. Установка свойства **Enabled** в значение **true** запускает таймер. Каждый тик таймера порождает событие **Tick**, обработчик которого обычно и создают в приложении. В этом обработчике могут изменяться какие либо величины, и вызваться принудительная перерисовка окна. Напоминаем, что вся отрисовка при создании анимации должна находиться в обработчике события **Paint**.

Создание анимации

Для создания простой анимации достаточно использовать таймер, при тике которого будут изменяться параметры изображения (например, координаты концов отрезка) и обработки события **Paint** для рисования по новым параметрам. При таком подходе не надо заботиться об удалении старого изображения (как в идеологии MS DOS), ведь оно создается в окне заново.

В качестве примера рассмотрим код анимации секундной стрелки часов:

```
using System;
using System.Collections.Generic; using System.ComponentModel; using
System.Data;
using System.Drawing; using System.Linq; using System.Text;
using System.Windows.Forms;
namespace WindowsFormsApplication1
{
    public partial class Form1 : Form
    {
        //описываем переменные доступные в любом обработчике событий
```

```

класс Form1 private int x1, y1, x2, y2, r;

    private double a;
    private Pen pen = new Pen(Color.DarkRed, 2);
    public Form1()
    {
        InitializeComponent();
    }
    private void Form1_Paint(object sender, PaintEventArgs e)
    {
        Graphics g = e.Graphics;
        g.DrawLine(pen, x1, y1, x2, y2); //рисуем секундную стрелку
    }
    private void Form1_Load(object sender, EventArgs e)
    {
        //определяем цент экрана x1 = ClientSize.Width / 2; y1 =
        ClientSize.Height / 2; r = 150; //задаем радиус
        a = 0; //задаем угол поворота
        //определяем конец часовой стрелки с учетом центра экрана
        x2 = x1 + (int) (r * Math.Cos(a)); y2 = y1 - (int) (r * Math.Sin(a));
    }
    private void timer1_Tick(object sender, EventArgs e)
    {
        a -= 0.1; //уменьшаем угол на 0,1 радиану
        //определяем конец часовой стрелки с учетом центра экрана
        x2 = x1 + (int)(r * Math.Cos(a)); y2 = y1 - (int)(r * Math.Sin(a));
        Invalidate(); //вынудительный вызов перерисовки (Paint)
    }
}

```

Выполнение индивидуального задания

Изучите с помощью справки MSDN методы и свойства классов **Graphics**,

Color, Pen и SolidBrush. Создайте собственное приложение выводющий на форму рисунок, состоящий из различных объектов (линий, многоугольников, эллипсов, прямоугольников и пр.), не закрашенных и закрашенных полностью. Используйте разные цвета и стили линий (сплошные, штриховые, штрих - пунктирные).

Создайте программу, показывающую пульсирующее сердце.

Создайте приложение, отображающее вращающийся винт самолета.

Разработайте программу анимации движущегося человечка.

Создайте программу, показывающую движение окружности по синусоиде.

Создайте приложение, отображающее движение окружности по спирали.

Разработайте программу анимации падения снежинки.

Создайте программу, показывающую скачущий мячик.

Создайте приложение, отображающее движение окружности вдоль границы окна. Учтите возможность изменения размеров окна.

Разработайте программу анимации летающего бумеранга.

Создайте программу, показывающую падение нескольких звезд одновременно.

Создайте приложение, отображающее хаотичное движение звезды в окне.

Разработайте программу анимации взлета ракеты. Старт осуществляется по нажатию специальной «красной» кнопки.

Создайте программу, показывающую движение окружности вдоль многоугольника. Число вершин вводится пользователем до анимации.

Создайте приложение, отображающее броуновское движение молекулы в окне.

Разработайте программу анимации движения планет в солнечной системе.

Создайте программу, показывающую движение квадратика по траектории, состоящей из 100 точек, и хранящихся в специальном массиве.

Создайте приложение, имитирующие механические часы.

Разработайте программу анимации падения несколько листков с дерева.

Движение не должно быть линейным.

Создайте программу, показывающую движение окружности по спирали с плавно изменяющейся скоростью.

Создайте приложение, отображающее движение автомобиля с вращающимися колесами.

ЛАБОРАТОРНАЯ РАБОТА № 11. Обработка изображений

Цель лабораторной работы: изучить возможности Visual Studio по открытию и сохранению файлов. Написать и отладить программу для обработки изображений.

Отображение графических файлов

Обычно для отображения точечных рисунков, рисунков из метафайлов, значков, рисунков из файлов в формате BMP, JPEG, GIF или PNG используется объект **PictureBox**, т.е. элемент управления **PictureBox** действует как контейнер для картинок. Можно выбрать изображение для вывода, присвоив значение свойству **Image**. Свойство **Image** может быть установлено в окне **Свойства** или в коде программы, указывая на рисунок, который следует отображать.

Элемент управления **PictureBox** содержит и другие полезные свойства, в том числе: **AutoSize** определяющее, будет ли изображение растянуто в элементе **PictureBox**, и **SizeMode**, которое может использоваться для растягивания, центрирования или увеличения изображения в элементе управления **PictureBox**.

Перед добавлением рисунка к элементу управления **PictureBox** в проект обычно добавляется файл рисунка в качестве ресурса. После добавления ресурса к проекту можно повторно использовать его. Например, может потребоваться отображение одного и того же изображения в нескольких местах.

Необходимо отметить, что поле **Image** само является классом для работы с изображениями, у которого есть свои методы. Например, метод **FromFile** используется для загрузки изображения из файла. Кроме класса **Image** существует класс **Bitmap**, который расширяет возможности класса **Image** за счет дополнительных методов для загрузки, сохранения и использования растровых изображений. Так метод **Save** класса **Bitmap** позволяет сохранять изображения в разных форматах, а методы **GetPixel** и

SetPixel позволяют получить доступ к отдельным пикселям рисунка.

Компоненты **OpenFileDialog** и **SaveFileDialog**

Компонент **OpenFileDialog** является стандартным диалоговым окном. Он

аналогичен диалоговому окну «Открыть файл» операционной системы Windows. Компонент OpenFileDialog позволяет пользователям просматривать папки личного компьютера или любого компьютера в сети, а также выбирать файлы, которые требуется открыть. Для вызова диалогового окна для выбора файла можно использовать метод ShowDialog() который возвращает true при корректном выборе.

Диалоговое окно возвращает путь и имя файла, который был выбран пользователем в специальном свойстве FileName.

Простой графический редактор

Создайте приложение, реализующее простой графический редактор. Функциями этого редактора должны быть: открытие рисунка, рисование поверх него простой кистью, сохранение рисунка в другой файл. Для этого создайте форму и разместите на ней элементы управления button и picturebox (рис 11.1).

В этом случае на понадобится из панели элементов размещать на форме компоненты диалоговых окон OpenFileDialog и SaveFileDialog. Эти элементы будут порождены динамически в ходе выполнения программы с помощью конструктора. Например так:

```
OpenFileDialog dialog = new OpenFileDialog();
```

Далее они будут вызывается с помощью метода ShowDialog().

Для кнопок «Открыть» и «Сохранить» создайте свои обработчики события. Также создайте обработчик события Load для формы. Для элемента управления picturebox1 создайте обработчики события MouseDown, MouseMove.

ЛАБОРАТОРНАЯ РАБОТА № 12. Методы

Цель лабораторной работы: научиться работать с методами, написать программу с использованием методов.

Общие понятия

Метод – это элемент класса, который содержит программный код.

Метод имеет следующую структуру:

[атрибуты] [спецификторы] тип имя ([параметры])

{

Тело метода;

}

Атрибуты – это особые указания компилятору на свойства метода.

Атрибуты используются редко.

Спецификаторы – это ключевые слова, предназначенные для разных целей, например:

Определяющие доступность метода для других классов:

private – метод будет доступен только внутри этого класса

protected – метод будет доступен также дочерним классам

public – метод будет доступен любому другому классу, который может получить доступ к данному классу

Указывающие доступность метода без создания класса

Задающие тип

Тип определяет результат, который возвращает метод: это может быть любой тип, доступный в C#, а также ключевое слово void, если результат не требуется.

Имя метода – это идентификатор, который будет использоваться для вызова метода. К идентификатору применяются те же требования, что и к именам переменных: он может состоять из букв, цифр и знака подчёркивания, но не может начинаться с цифры. Параметры – это список переменных, которые можно передавать в метод при вызове. Каждый параметр состоит из типа и названия

переменной. Параметры разделяются запятой.

Тело метода – это обычный программный код, за исключением того, что он не может содержать определения других методов, классов, пространств имён и т. д. Если метод должен возвращать какой-то результат, то обязательно в конце должно присутствовать ключевое слово `return` с возвращаемым значением. Если возвращение результатов не нужно, то использование ключевого слова `return` не обязательно, хотя и допускается.

Параметры по умолчанию

Язык C# начиная с версии 4.0 (Visual Studio 2010) позволяет задавать некоторым параметрам значения по умолчанию – так, чтобы при вызове метода можно было опускать часть параметров.

В первом случае параметр `Optional` будет равен 20, так как он явно задан, а во втором будет равен 5, т.к. явно он не задан, и компилятор берёт значение по умолчанию.

Параметры по умолчанию можно ставить только в правой части списка параметров, например, такая сигнатура метода компилятором принята не будет:

```
private void GetData(int Optional = 5, int Number)
```

Передача параметров по значению и по ссылке

Когда параметры передаются в метод обычным образом (без дополнительных ключевых слов `ref` и `out`), любые изменения параметров внутри метода не влияют на его значение в основной программе. Предположим, у нас есть следующий метод:

```
private void Calc(int Number)
{
    Number = 10;
}
```

Видно, что внутри метода происходит изменение переменной `Number`, которая была передана как параметр. Попробуем вызвать метод:

```
int n = 1; Calc(n);
```

```
Console.WriteLine(n);
```

На экране появится число 1, то есть, не смотря на изменение переменной в методе Calc, значение переменной в главной программе не изменилось. Это связано с тем, что при вызове метода создаётся *копия* переданной переменной, именно её изменяет метод. При завершении метода значение копий теряется. Такой способ передачи параметра называется *передачей по значению*.

Чтобы метод мог изменять переданную ему переменную, её следует передавать с ключевым словом `ref` — оно должно быть как в сигнатуре метода, так и при вызове:

```
private void Calc(ref int Number)
{
    Number = 10;
}
int n = 1; Calc(ref n);
Console.WriteLine(n);
```

В этом случае на экране появится число 10: изменение значения в методе сказалось и на главной программе. Такая передача метода называется *передачей по ссылке*, т.е. передаётся уже не копия, а ссылка на реальную переменную в памяти.

Если метод использует переменные по ссылке только для возврата значений и ему не важно, что в них было изначально, то можно не инициализировать такие переменные, а передавать их с ключевым словом `out`. Компилятор понимает, что начальное значение переменной не важно и не ругается на отсутствие инициализации:

```
private void Calc(out int Number)
{
    Number = 10;
}
int n; // Ничего не присваиваем!
Calc(out n);
```

ЛАБОРАТОРНАЯ РАБОТА № 13. Сортировка

Цель лабораторная работа: освоить основные алгоритмы сортировки, написать программу с использованием этих алгоритмов.

Общие понятия

Сортировка – это процесс упорядочения элементов массива или списка по возрастанию или убыванию.

Существует много алгоритмов сортировки, отличающихся по ряду характеристик:

Время работы, или вычислительная сложность – количество операций, затрачиваемых алгоритмом. Обычно оценивается худший сценарий, когда исходный массив оказывается максимально не упорядочен с точки зрения алгоритма.

Затрачиваемая память (помимо исходного массива) – некоторые алгоритмы требуют выделения дополнительной памяти для временного хранения данных или формирования нового выходного массива.

Кроме того, алгоритмы можно разделить по типу доступа к данным:

Алгоритмы *внутренней сортировки* применяются для сортировки данных, целиком находящихся в оперативной памяти.

Алгоритмы *внешней сортировки* оперируют данными, не помещающимися в оперативную память. Такие алгоритмы используют внешнюю память, доступ к которой требует существенно большего времени, поэтому требуются специальные алгоритмические решения, чтобы каждый элемент использовался алгоритмом минимальное количество раз.

Алгоритмы сортировки. Метод пузырька

Данный алгоритм является достаточно простым и поэтому получил широкое распространение. Вычислительная сложность алгоритма квадратичная – $O(n^2)$, поэтому алгоритм эффективен только на небольших массивах данных.

Алгоритм проходит все элементы массива и попарно сравнивает их друг с другом. Если порядок сравниваемых элементов неверный, алгоритм меняет

элементы местами:

```
// Сортировка пузырьком
void BubbleSort(ref int[] Array)
{
    // Перебираем все элементы массива (кроме последнего)
    for (int i = 0; i < Array.Length - 1; i++)
        // Перебираем все элементы справа от i
        for (int j = i + 1; j < Array.Length; j++)
            // Правильный ли порядок элементов?
            if (Array[i] > Array[j])
                {
                    // Нет - меняем порядок
                    int t = Array[i]; Array[i] =
Array[j]; Array[j] = t;
                }
}
```

Сортировка выбором

Сортировка выбором имеет квадратичную сложность $O(n^2)$ и, как и предыдущий метод пузырька, эффективен лишь на небольших объемах данных.

Алгоритм находит номер минимального значения в текущем списке, меняет этот элемент со значением первой неотсортированной позиции (если минимальный элемент не находится на данной позиции), а затем сортирует хвост списка, исключив из рассмотрения уже отсортированные элементы:

```
// Сортировка выбором
void SelectionSort(ref int[] Array)
{
    // Перебираем все элементы массива (кроме последнего)
    // i - позиция первого неотсортированного элемента
    for (int i = 0; i < Array.Length - 1; i++)
    {
```

```

// Позиция минимального элемента справа от i int min = i;
// Перебираем все элементы справа от i
for (int j = i + 1; j < Array.Length; j++)
// Меньше ли очередной элемент минимального?
if (Array[j] < Array[min])
min = j; // Да - теперь это новый минимальный элемент
// Минимальный элемент не на первом месте? Меняем местами!
if (min != i)
{
int t = Array[i]; Array[i] = Array[min]; Array[min] = t;
}
}
}

```

Быстрая сортировка

Алгоритм быстрой сортировки является одним из самых быстрых алгоритмов сортировки: в лучшем случае он имеет логарифмическую сложность, в худшем — квадратичную. Алгоритм выполняется следующим образом:

Выбирается некоторый элемент, который называется *опорным*.

Реорганизуем массив таким образом, чтобы все элементы, меньшие или равные опорному элементу, оказались слева от него, а все элементы, большие опорного — справа от него.

Рекурсивно упорядочиваем массивы, лежащие слева и справа от опорного элемента.

```

// Быстрая сортировка
void QuickSort(ref int[] Array, int Left, int Right)
{
// i и j - индексы границ разделяемого массива
int i = Left; int j = Right;
// x - индекс опорного элемента

```

```

int x = Array[(Left + Right) / 2]; do
{
// Ищем элемент слева, который больше опорного
while (Array[i] < x)
++i;
// Ищем элемент справа, который меньше опорного
while (Array[j] > x)
--j;
// Если индексы не поменялись местами, то обмениваем
элементы
if (i <= j)
{
int t = Array[i]; Array[i] = Array[j]; Array[j] = t;
i++; j--;
}

справа

}
} while (i <= j);
// Рекурсивно выполняем быструю сортировку для массивов
слева и

```

```
if (Left < j)
QuickSort(ref Array, Left, j); if (i < Right)
QuickSort(ref Array, i, Right);Поиск элемента
```

Алгоритмы поиска позволяют найти индекс элемента с требуемым значением.

Если массив не упорядочен, то возможен лишь *простой поиск*: перебор всех элементов массива до тех пор, пока не встретится элемент с нужным значением или не закончится массив. Если элемент найден, поиск должен быть прекращён, поскольку дальнейший просмотр массива не имеет смысла:

```
// Простой поиск элемента в массиве
int IndexOf(ref int[] Array, int Value)
{
// Перебираем все элементы массива
for (int i = 0; i < Array.Length; i++)
// Если нашли нужное значение, то возвращаем его индекс
if (Array[i] == Value) return i;
// Перебор закончился безрезультатно - возвращаем -1 return
-1;
}
```

Если алгоритм поиска не нашёл подходящий элемент, он должен каким-то образом сигнализировать об этом вызывающей программе. Чаще всего в таком случае возвращается значение -1 — число, которое заведомо не может использоваться в качестве индекса массива.

Вычислительная сложность алгоритма простого поиска — линейная $O(n)$.

Если же массив упорядочен по возрастанию, то возможно использование дихотомического рекурсивного алгоритма: массив каждый раз делится пополам и если искомый элемент меньше среднего, то поиск продолжается в левой его половине, иначе — в правой:

```
// Дихотомический поиск элемента в массиве
```

```

    static int IndexOf(ref int[] Array, int Value, int Left,
int Right)
    {
        // Находим середину диапазона
        int x = (Left + Right) / 2;
        // Если нашли значение - возвращаем его индекс
        if (Array[x] == Value) return x;
        // Если середина совпадает с левой или правой границами -
значение не найдено
        if ((x == Left) || (x == Right)) return -1;
        // Продолжаем поиск слева или справа от середины
        if (Array[x] < Value)
            return IndexOf(ref Array, Value, x, Right); else
            return IndexOf(ref Array, Value, Left, x);
    }

```

Вычислительная сложность алгоритма – логарифмическая.

Выполнение индивидуального задания

Общая часть задания: сформировать массив из 100 случайных чисел. Выполнить простой поиск элемента, подсчитать количество итераций. Отсортировать массив методом, указанным в своём варианте. Выполнить поиск элемента методом дихотомии, подсчитать количество итераций. Сделать выводы.

Метод пузырька

Сортировка выбором

Быстрая сортировка

Список рекомендуемых источников

Основная литература

1. Платонов, В. В. *Программно-аппаратные средства защиты информации* : учебник / В. В. Платонов, М. А. Ушаков. — Москва : Академия, 2020. — 288 с. — ISBN 978-5-4468-5678-9.
2. Тарасова, С. В. Методические указания по программированию на Python для задач информационной безопасности / С. В. Тарасова, Е. К. Баранова. — Москва : КноРус, 2021. — 180 с. — ISBN 978-5-406-08712-3.

Дополнительная литература

1. Галатенко, В. А. Основы информационной безопасности : учебное пособие / В. А. Галатенко. — Москва : Интернет-Университет Информационных Технологий (ИНТУИТ), 2016. — 320 с. — ISBN 978-5-9963-0421-9.
2. Складов, Д. В. Программирование и защита информации : учебное пособие / Д. В. Складов. — Санкт-Петербург : Лань, 2019. — 256 с. — ISBN 978-5-8114-3412-3.
3. Шаньгин, В. Ф. Информационная безопасность компьютерных систем и сетей : учебное пособие / В. Ф. Шаньгин. — Москва : Форум, 2018. — 416 с. — ISBN 978-5-8199-0789-1.