МИНИСТЕРСТВО ОБРАЗОВАНИЯ И НАУКИ РОССИЙСКОЙ ФЕДЕРАЦИИ
Федеральное государственное бюджетное образовательное учреждение
высшего профессионального образования
**«НАЦИОНАЛЬНЫЙ ИССЛЕДОВАТЕЛЬСКИЙ
ТОМСКИЙ ПОЛИТЕХНИЧЕСКИЙ УНИВЕРСИТЕТ»**

А.Ю. Дёмин, В.А. Дорофеев

ЛАБОРАТОРНЫЙ ПРАКТИКУМ ПО ИНФОРМАТИКЕ

*Рекомендовано в качестве учебного пособия
Редакционно-издательским советом
Томского политехнического университета*

Издательство
Томского политехнического университета
2014

УДК 004.43(076.5)
ББК 32.973я73
Д30

Дёмин А.Ю.

Д30 Лабораторный практикум по информатике: учебное пособие / А.Ю. Дёмин, В.А. Дорофеев; Томский политехнический университет. − Томск: Изд-во Томского политехнического университета, 2014. – 132 с.

Пособие знакомит с языком программирования C#. Рассмотрены основные конструкции языка и типы данных; среда разработки Visual Studio 2010; работа с базовыми элементами управления. Содержаться указания и задания для выполнения лабораторных работ.

Предназначено для студентов, обучающихся по направлениям 210700 «Инфокоммуникационные технологии и системы связи», 220400 «Управление в технических системах», 220700 «Автоматизация технологических процессов и производств», 221000 «Мехатроника и робототехника», 222000 «Инноватика», 230100 «Информатика и вычислительная техника».

УДК 004.43(076.5)
ББК 32.973я73

Рецензенты

Доктор технических наук
профессор кафедры комплексной информационной
безопасности электронно-вычислительных систем ТУСУРа
Р.В. Мещеряков

Кандидат технических наук
доцент кафедры программирования ТГУ
О.И. Голубева

© ФГБОУ ВПО НИ ТПУ, 2014
© Дёмин А.Ю., Дорофеев В.А., 2014
© Оформление. Издательство Томского политехнического университета, 2014

ОГЛАВЛЕНИЕ

ЛАБОРАТОРНАЯ РАБОТА № 1.
Изучение среды разработки Visual Studio 6
1.1. Интегрированная среда разработчика Visual Studio 6
1.2. Настройка формы 8
1.3. Размещение элементов управления на форме 9
1.4. Размещение строки ввода 9
1.5. Размещение надписей 10
1.6. Написание программы обработки события 11
1.7. Написание программы обработки события нажатия кнопки 11
1.8. Написание программы обработки события загрузки формы 11
1.9. Запуск и работа с программой 13
1.10. Динамическое изменение свойств 14
1.11. Выполнение индивидуального задания 14
Индивидуальные задания 14
ЛАБОРАТОРНАЯ РАБОТА № 2. Линейные алгоритмы 18
2.1. Структура приложения 18
2.2. Работа с проектом 20
2.3. Описание данных 20
Целочисленные типы 21
Типы с плавающей точкой 21
Символьные типы 22
Логический тип данных 22
2.4. Ввод/вывод данных в программу 23
2.5. Арифметические действия и стандартные функции 24
2.6. Пример написания программы 25
2.7. Выполнение индивидуального задания 27
Индивидуальные задания 28
ЛАБОРАТОРНАЯ РАБОТА № 3. Разветвляющиеся алгоритмы 31
3.1. Логические переменные и операции над ними 31
3.2. Условные операторы 31
3.3. Кнопки-переключатели 35
3.4. Пример написания программы 36
3.4.1. Создание формы 36
3.4.2. Создание обработчиков событий 36
Индивидуальные задания 37

ЛАБОРАТОРНАЯ РАБОТА № 4. Циклические алгоритмы ... 39
4.1. Операторы организации циклов ... 39
4.2. Цикл с предусловием ... 39
4.3. Цикл с постусловием ... 40
4.4. Цикл с параметром ... 40
4.5. Средства отладки программ ... 41
4.6. Порядок выполнения задания ... 43
Индивидуальные задания ... 44
ЛАБОРАТОРНАЯ РАБОТА № 5. Классы и объекты ... 46
5.1. Классы и объекты ... 46
5.2. Динамическое создание объектов ... 46
5.3. Область видимости ... 48
5.4. Операции is и as ... 49
5.5. Сведения, передаваемые в событие ... 50
Индивидуальные задания ... 50
ЛАБОРАТОРНАЯ РАБОТА № 6. Строки ... 53
6.1. Строковый тип данных ... 53
6.2. Более эффективная работа со строками ... 53
6.3. Элемент управления ListBox ... 54
6.4. Порядок выполнения индивидуального задания ... 55
Индивидуальные задания ... 56
ЛАБОРАТОРНАЯ РАБОТА № 7. Одномерные массивы ... 58
7.1. Работа с массивами ... 58
7.2. Случайные числа ... 60
7.3. Порядок выполнения индивидуального задания ... 60
Индивидуальные задания ... 62
ЛАБОРАТОРНАЯ РАБОТА № 8. Многомерные массивы ... 64
8.1. Двухмерные массивы ... 64
8.2. Элемент управления DataGridView ... 64
8.3. Порядок выполнения задания ... 65
Индивидуальные задания ... 66
ЛАБОРАТОРНАЯ РАБОТА № 9. Графики функций ... 69
9.1. Как строится график с помощью элемента управления Chart ... 69
9.2. Пример написания программы ... 70
9.3. Выполнение индивидуального задания ... 71
ЛАБОРАТОРНАЯ РАБОТА № 10. Компьютерная графика ... 72
10.1. Событие Paint ... 72

10.2. Объект Graphics для рисования 72
10.3. Методы и свойства класса Graphics 73
Индивидуальное задание 77
ЛАБОРАТОРНАЯ РАБОТА № 11. Анимация 78
11.1. Работа с таймером 78
11.2. Создание анимации 78
11.3. Движение по траектории 79
Индивидуальное задание 81
ЛАБОРАТОРНАЯ РАБОТА № 12. Обработка изображений 82
12.1. Отображение графических файлов 82
12.2. Элементы управления OpenFileDialog и SaveFileDialog 83
12.3. Простой графический редактор 83
Индивидуальное задание 87
ЛАБОРАТОРНАЯ РАБОТА № 13. Методы 89
13.1. Общие понятия 89
13.2. Перегрузка методов 90
13.3. Параметры по умолчанию 91
13.4. Передача параметров по значению и по ссылке 92
Индивидуальное задание 93
ЛАБОРАТОРНАЯ РАБОТА № 14. Рекурсия 95
14.1. Общие понятия 95
14.2. Формирование задержки с помощью таймера 98
Индивидуальное задание 99
ЛАБОРАТОРНАЯ РАБОТА № 15. Сортировка и поиск 105
15.1. Общие понятия 105
15.2. Алгоритмы сортировки. Метод пузырька 105
15.3. Сортировка выбором 106
15.4. Быстрая сортировка 107
15.5. Поиск элемента 108
Индивидуальное задание 109
ИНДИВИДУАЛЬНЫЕ ЗАДАНИЯ ПОВЫШЕННОЙ СЛОЖНОСТИ 110
ПРИЛОЖЕНИЕ 1. Свойства элементов управления 121
ПРИЛОЖЕНИЕ 2. События элементов управления 124
ПРИЛОЖЕНИЕ 3. Методы для работы со строками 128
ПРИЛОЖЕНИЕ 4. Методы для работы с массивами 130
СПИСОК ЛИТЕРАТУРЫ 131

ЛАБОРАТОРНАЯ РАБОТА № 1.
ИЗУЧЕНИЕ СРЕДЫ РАЗРАБОТКИ VISUAL STUDIO

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

1.1. 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).

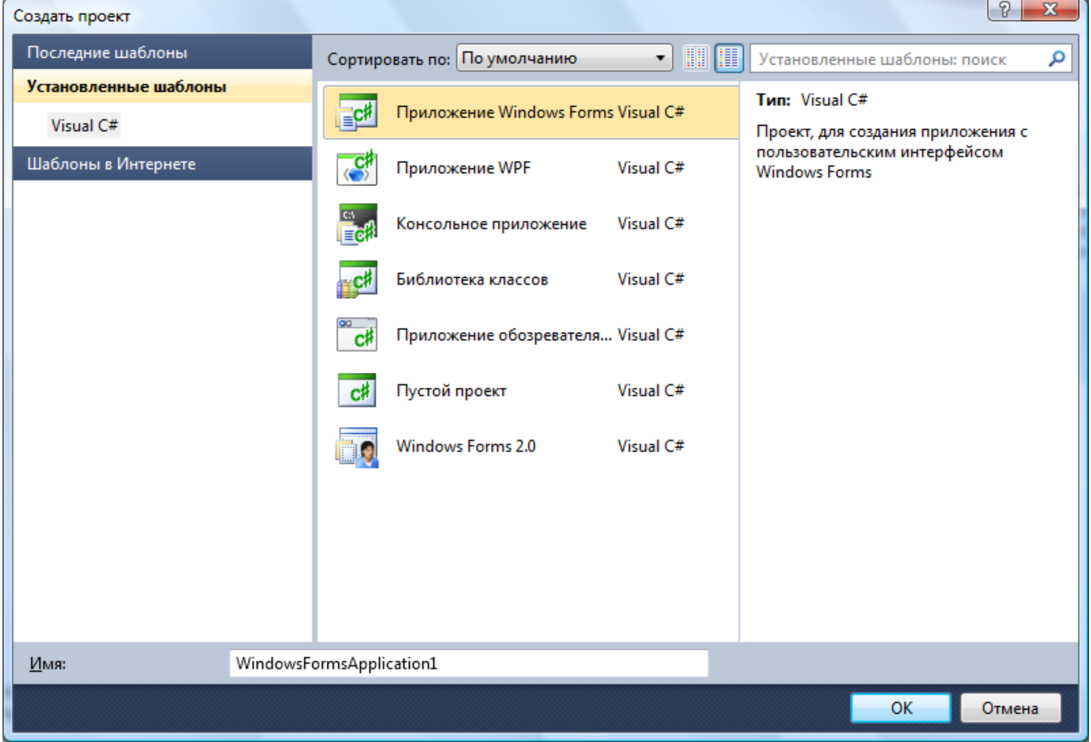


Рис. 1.1. Диалог создания нового проекта

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:

1. *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.

2. *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 OK. Через несколько секунд Visual Studio создаст проект и Вы сможете увидеть на экране картинку, подобную представленной на рис. 1.2.

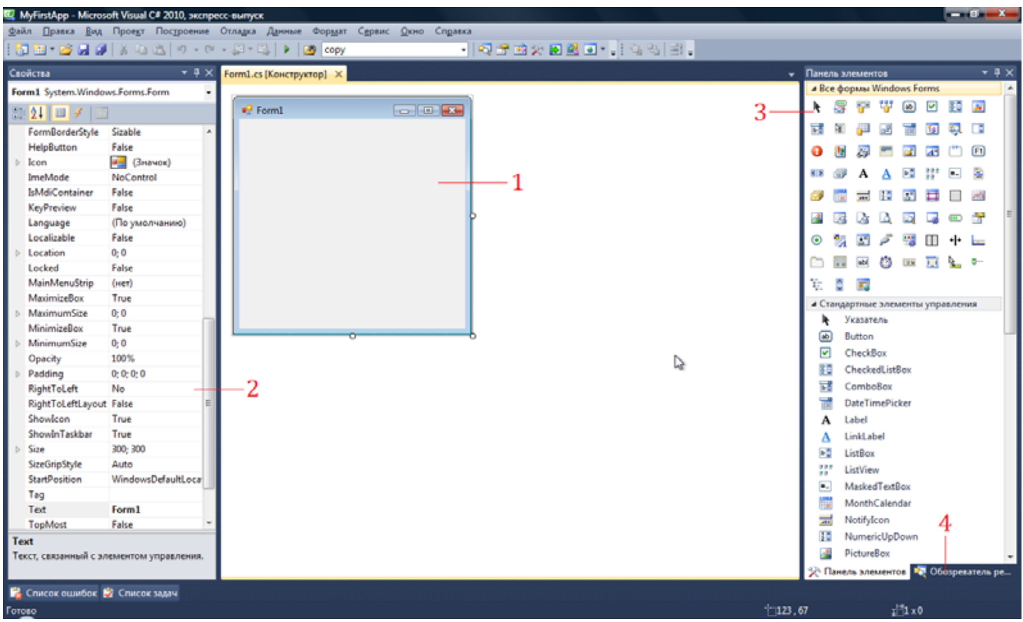


Рис. 1.2. Главное окно Visual Studio

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. Чтобы было удобнее искать нужные свойства, можно отсортировать их по алфавиту с помощью кнопки . Если этого окна на экране нет, его можно активировать в меню *Вид → Окно свойств* (иногда этот пункт вложен в подпункт *Другие окна*).

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, ее 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 → Обозреватель решений*.

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

Окно текста программы предназначено для просмотра, написания и редактирования текста программы. Переключаться между формой и текстом программы можно с помощью команд *Вид → Код* и *Вид → Конструктор*. 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.

1.2. 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 щелкните кнопкой мыши на форме, в окне свойств найдите и щелкните мышкой 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`.

1.3. 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*.

Щелкните на элементе управления, который хотите добавить, а затем щелкните 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.

1.4. Размещение строки ввода

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 будет вводиться имя пользователя.

Выберите на панели элементов пиктограмму с названием `TextBox`, щелкните мышью в том месте формы, где вы хотите ее поставить. Захватив его мышкой, отрегулируйте размеры элемента управления и его положение. Обратите внимание на то, что теперь в тексте программы можно использовать переменную `textBox1`, которая соответствуют добавленному элементу управления. В этой переменной в свойстве `Text` будет содержаться строка символов (тип `string`) и отображаться в соответствующем окне `TextBox`.

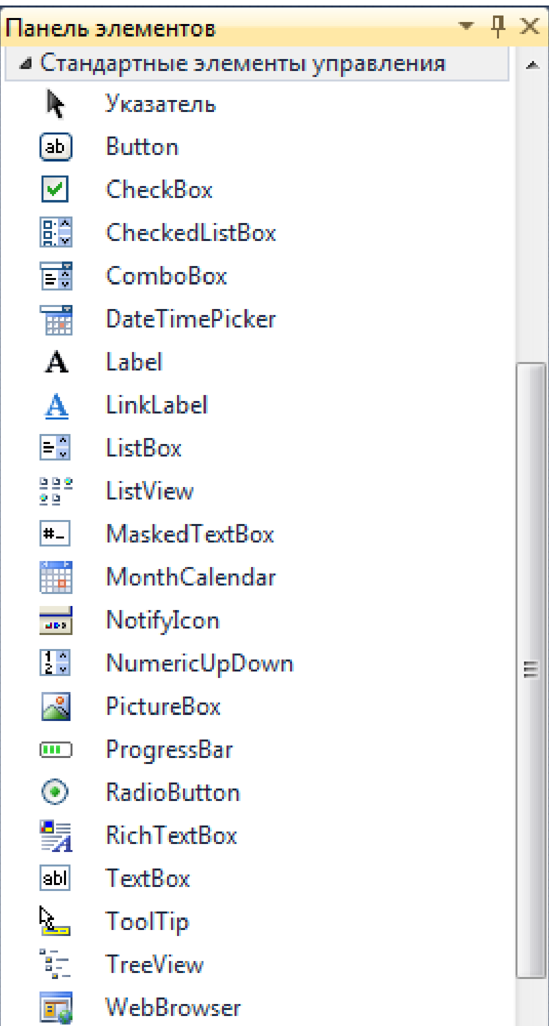


Рис. 1.3. Панель элементов

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

1.5. Размещение надписей

На форме могут размещаться пояснительные надписи. Для нанесения таких надписей на форму используется элемент управления `Label`. Выберите на панели элементов пиктограмму с названием `Label`, щелкните на ней мышью. После этого в нужном месте формы щелкните мышью, появится надпись `label1`. Щелкнув на ней мышью, отрегулируйте размер и, изменив свойство `Text` в окне свойств, введите строку, например «*Введите свое имя:*», а также выберите размер символов (свойство `Font`).

Обратите внимание, что в тексте программы теперь можно обращаться к новой переменной типа `Label`. В ней хранится пояснительная строка, которую можно изменять в процессе работы программы.

1.6. 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 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 создается обработчик события нажатия.

1.7. Написание программы обработки события нажатия кнопки

Поместите на форму кнопку, которая описывается элементом управления `Button`. С помощью окна свойств измените заголовок (`Text`) на слово «*Привет*» 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 e)
{

}
```

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

```
MessageBox.Show("Привет, " + textBox1.Text + "!");
```

1.8. Написание программы обработки события загрузки формы

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. Выделите форму щелчком по ней, чтобы вокруг нее появилась рамка из точек. В окне свойств найдите событие `Load`. Щелкнете по данной строчке дважды мышкой. Появится метод:

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

```
}
```

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 описаны ниже:

- `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 `KeyDown` 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.

Важное примечание! Если какой-то обработчик был добавлен по ошибке или больше не нужен, то для его удаления нельзя просто удалить программный код обработчика! Сначала нужно удалить строку с именем обработчика в окне свойств на закладке ⚡. В противном случае программа может перестать компилироваться и даже отображать форму в дизайнере Visual Studio.

1.9. 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.

Если в программе есть ошибки, то в окне Список ошибок появятся найденные ошибки, а программа обычно не запускается. Иногда, однако, может быть запущена предыдущая версия программы, в которой еще нет последних изменений: чтобы этого не происходило, нужно в настройках Visual Studio установить опцию показа окна ошибок и запретить запуск при наличии ошибок (рис. 1.4 и 1.5).

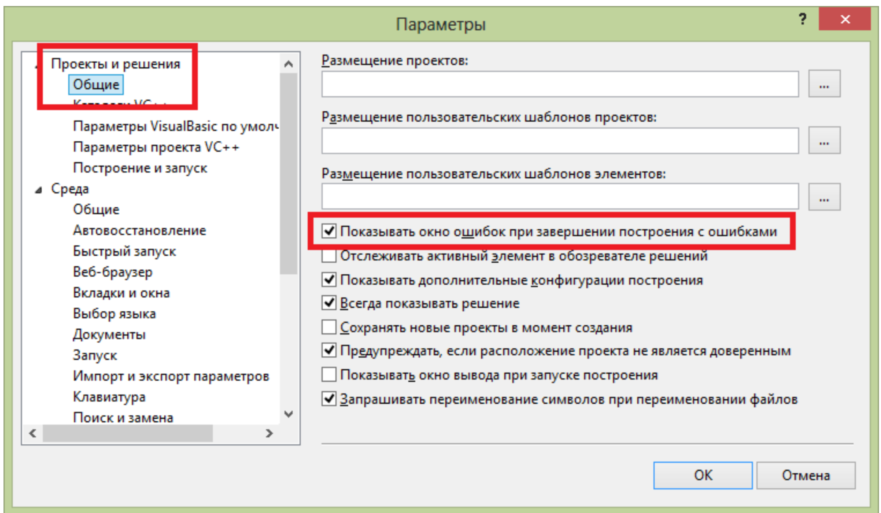


Рис. 1.4. Опция показа сообщений об ошибках

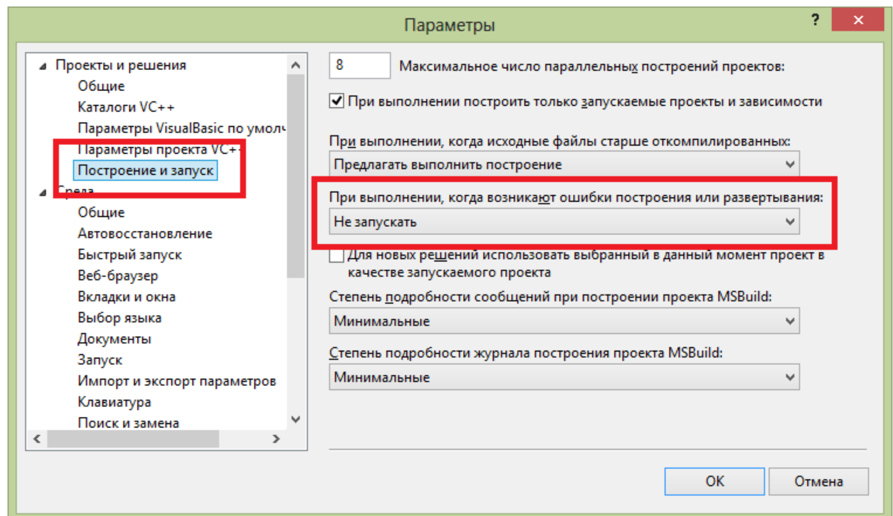


Рис. 1.5. Опция отключения запуска предыдущей версии при ошибках построения

Для завершения работы программы и возвращения в режим проектирования формы не забудьте закрыть окно программы!

1.10. 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., которые могут быть заменены 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.

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

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

1.11. Выполнение индивидуального задания

По указанию преподавателя выберите свое индивидуальное задание. Уточните условие задания, количество, наименование, типы исходных данных. Прочтите в Приложении 1 описание свойств и описание элементов управления `Form`, `Label`, `TextBox`, `Button`. С помощью окна свойств установите первоначальный цвет формы, шрифт выводимых символов.

Индивидуальные задания

1. Разместите на форме четыре кнопки (`Button`). Сделайте на кнопках следующие надписи: «красный», «зеленый», «синий», «желтый». Создайте четыре обработчика события нажатия на данные кноп-

ки, которые будут менять цвет формы в соответствии с текстом на кнопках.

2. 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, написанные на кнопках. Создайте обработчик события создания формы (`Load`), который будет устанавливать цвет формы и менять текст метки на строку «Начало работы».

3. Разместите на форме ряд кнопок (`Button`) напротив каждой поле ввода (`TextBox`) и одну метку (`Label`). Создайте обработчики события нажатия на данные кнопки, которые будут менять текст в метке. Текст в метке берется из поля ввода напротив нажимаемой кнопки.

4. Разместите на форме ряд кнопок (`Button`), и одно поле ввода (`TextBox`). Создайте обработчики события нажатия на данные кнопки, которые будут менять текст на нажатой кнопке. Текст на кнопке берется из поля ввода.

5. Разместите на форме ряд кнопок (`Button`) и ряд меток (`Label`). Создайте обработчики события нажатия на данные кнопки, которые будут менять цвет двух меток. Создайте обработчик события нажатия кнопки мыши на форме (`Click`), который будет устанавливать цвет всех меток в белый.

6. Разместите на форме ряд кнопок (`Button`) и ряд меток (`Label`). Создайте обработчик события создания формы (`Load`), который будет делать все метки невидимыми. Создайте обработчики события нажатия на кнопки, которые будут менять свойство метки `Visible`, тем самым делать их видимыми.

7. Разместите на форме ряд кнопок (`Button`), напротив каждой поле ввода (`TextBox`). Создайте обработчики события нажатия на данные кнопки, которые будут менять заголовок окна. Текст в заголовке берется из поля ввода напротив нажимаемой кнопки.

8. 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».

9. 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.

10. 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 «+++++».

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

12. Разработайте игру, которая заключается в следующем. На форме размещены четыре кнопки (Button) и четыре метки (Label). При нажатии на кнопку часть надписей становится невидимой, а часть, наоборот, становится видимой. Цель игры – скрыть все надписи.

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

14. 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.

15. 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».

16. 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 изменения размера формы (Resize), который будет очищать все поля ввода.

17. 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 мышки на форме (Click), который будет устанавливать цвет формы и менять текст метки на строку «Начало работы» и очищать поле ввода.

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

19. Реализуйте игру минер на поле 3×3 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.

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

ЛАБОРАТОРНАЯ РАБОТА № 2. ЛИНЕЙНЫЕ АЛГОРИТМЫ

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

2.1. 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, а уже в ней – 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*. Примером другого типа проекта является 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 имен. Каждое пространство имен включает в себя классы, выполняющие определенную 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 имен.

Для того чтобы не происходило конфликтов имен 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`.

2.2. Работа с проектом

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 все*. 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 `.sln` и сделать на нем двойной щелчок.

2.3. Описание данных

Типы данных имеют особенное значение в C#, поскольку это *строго типизированный язык*. Это означает, что все операции подвергаются строгому контролю со стороны компилятора на соответствие типов, 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, допустимая в ряде скриптовых языков, в C# вообще не существует. Более того, тип зна-

чения определяет те операции, которые разрешается выполнять над ним. Операция, разрешенная для одного типа данных, может оказаться недопустимой для другого.

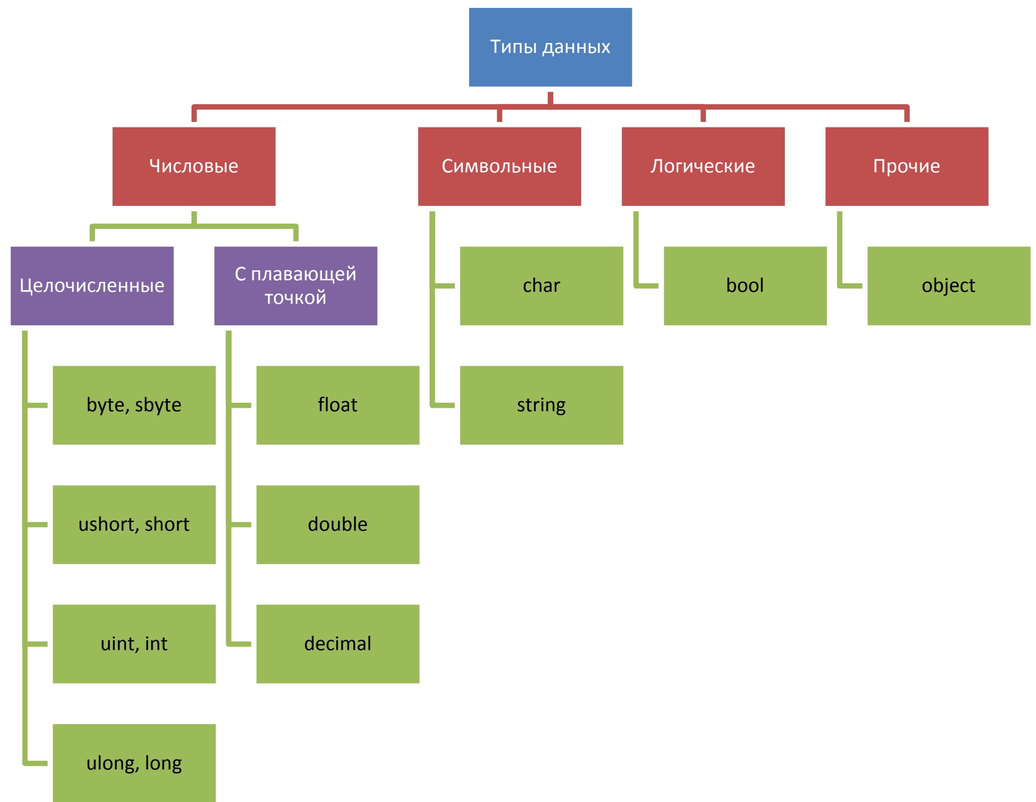


Рис. 2.1. Структура типов данных

Целочисленные типы

В C# определены девять целочисленных типов: `char`, `byte`, `sbyte`, `short`, `ushort`, `int`, `uint`, `long` и `ulong`. Тип `char` может хранить числа, но чаще используется для представления символов. Остальные восемь целочисленных типов предназначены для числовых расчетов.

Некоторые целочисленные типы могут хранить как положительные, так и отрицательные значения (`sbyte`, `short`, `int` и `long`), другие же – только положительные (`char`, `byte`, `ushort`, `uint` и `ulong`).

Типы с плавающей точкой

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

ного для типов float и double. Вычисления с использованием этого типа выполняются программно и поэтому более медленны.

Числа, входящие в выражения, C# по умолчанию считает целочисленными. Поэтому следующее выражение будет иметь значение 0, ведь если 1 нацело разделить на 2, то получится как раз 0:

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

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

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

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

```
2.5e-2
```

Символьные типы

В C# символы представлены не 8-разрядным кодом, как во многих других языках программирования, 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.

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

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.

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 задается *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
```

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
```

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;
```

2.4. 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 `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;
    }
```

> В языках программирования в дробных числах чаще всего используется точка, например: «15.7». Однако в C# методы преобразования строк в числа (вроде `double.Parse()` или `Convert.ToFloat()`) учитывают региональные настройки Windows, в которых в качестве десятичной точки используется символ *запятой* (например, «15,7»). Поэтому в полях `TextBox` в формах следует вводить дробные числа с *запятой*, а не с точкой. В противном случае преобразование не выполнится, а программа остановится с ошибкой.

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();
    }
```

2.5. Арифметические действия и стандартные функции

При вычислении выражения, стоящего в правой части оператора присвоения, могут использоваться арифметические операции:

- умножение (×);
- сложение (+);
- вычитание (–);
- деление (/);
- остаток от деления (%).

Для задания приоритетов операций могут использоваться круглые скобки `( )`. Также могут использоваться стандартные математические функции, представленные методами класса `Math`:

- `Math.Sin(a)` – синус;
- `Math.Sinh(a)` – гиперболический синус;

- `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.

В тригонометрических функциях все аргументы задаются в радианах.

2.6. Пример написания программы

З а д а н и е : составить программу вычисления для заданных значений x, y, z арифметического выражения:

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

Панель диалога программы организовать в виде, представленном на рис. 2.2.

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.

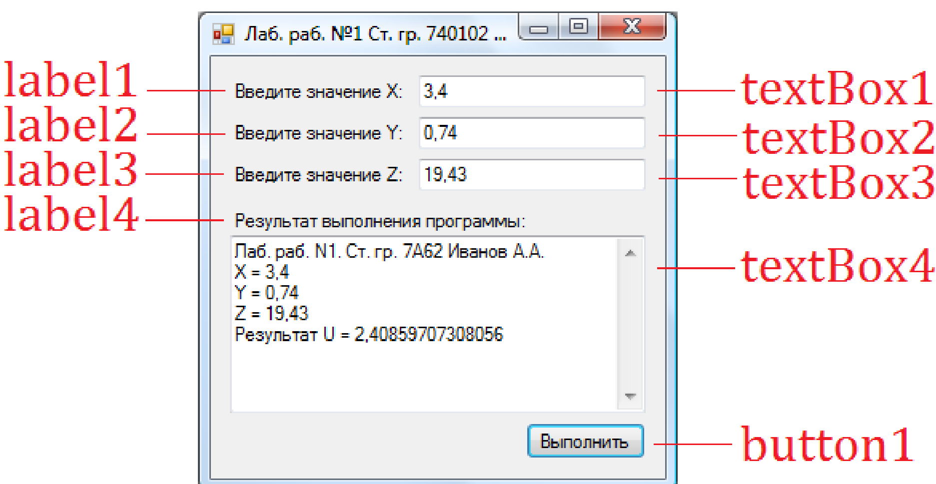


Рис. 2.2. Внешний вид программы

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)
        {
            // Начальное значение X
            textBox1.Text = "3,4";
            // Начальное значение Y
            textBox2.Text = "0,74";
            // Начальное значение Z
            textBox3.Text = "19,43";
        }
```

```
            private void button1_Click(object sender,
                EventArgs e)
            {
                // 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*z));
                double u = a -b * c;
                // Выводим результат в окно
                textBox4.Text += Environment.NewLine +
                    "Результат U = " + u.ToString();
            }
        }
    }
```

Если просто скопировать этот текст и заменить им то, что было в редакторе кода Visual Studio, то программа не заработает. Правильнее будет создать обработчики событий `Load` у формы и `Click` у кнопки и уже в них вставить соответствующий код. Это замечание относится и ко всем последующим лабораторным работам.

2.7. Выполнение индивидуального задания

Ниже приведено 20 вариантов задач. По указанию преподавателя выберите свое индивидуальное задание. Уточните условие задания, количество, наименование, типы исходных данных. В соответствии с этим установите необходимое количество окон `TextBox`, тексты заголовков на форме, размеры шрифтов, а также типы переменных и функции преобразования при вводе и выводе результатов. Для проверки правильности

программы после задания приведен контрольный пример: тестовые значения переменных, используемых в выражении, и результат, который при этом получается.

Индивидуальные задания

1. $$t = \frac{2\cos\left(x - \frac{\pi}{6}\right)}{0.5 + \sin^2 y}\left(1 + \frac{z^2}{3 - z^2/5}\right).$$

   При $x = 14.26$, $y = -1.22$, $z = 3.5\times10^{-2}$ $t = 0.564849$.

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

   При $x = -4.5$, $y = 0.75\times10^{-4}$, $z = 0.845\times10^2$ $u = -55.6848$.

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

   При $x = 3.74\times10^{-2}$, $y = -0.825$, $z = 0.16\times10^2$, $v = 1.0553$.

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

   При $x = 0.4\times10^4$, $y = -0.875$, $z = -0.475\times10^{-3}$ $w = 1.9873$.

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

   При $x = -15.246$, $y = 4.642\times10^{-2}$, $z = 20.001\times10^2$ $\alpha = -182.036$.

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

   При $x = 16.55\times10^{-3}$, $y = -2.75$, $z = 0.15$ $\beta = -38.902$.

7. $$\gamma = 5\,\mathrm{arctg}(x) - \frac{1}{4}\arccos(x)\frac{x + 3|x-y| + x^2}{|x-y|z + x^2}.$$

   При $x = 0.1722$, $y = 6.33$, $z = 3.25\times10^{-4}$ $\gamma = -172.025$.

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

   При $x = -2.235\times10^{-2}$, $y = 2.23$, $z = 15.221$ $\varphi = 39.374$.

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

При $x = 1.825\times10^2$, $y = 18.225$, $z = –3.298\times10^{-2}$ $\psi = 1.2131$.

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

При $x = 3.981\times10^{-2}$, $y = –1.625\times10^3$, $z = 0.512$ $a = 1.26185$.

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

При $x = 6.251$, $y = 0.827$, $z = 25.001$ $b = 0.7121$.

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

При $x = 3.251$, $y = 0.325$, $z = 0.466\times10^{-4}$ $c = 4.025$.

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

При $x = 17.421$, $y = 10.365\times10^{-3}$, $z = 0.828\times10^5$ $f = 0.33056$.

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

При $x = 12.3\times10^{-1}$, $y = 15.4$, $z = 0.252\times10^3$ $g = 82.8257$.

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

При $x = 2.444$, $y = 0.869\times10^{-2}$, $z = –0.13\times10^3$ $h = –0.49871$.

16. $y = \sqrt{cx} - 2.7\dfrac{|c|+|x|}{c^2x^2}\cdot e^{cx} + \cos\dfrac{(a+b)^2}{cx-b}$

$a = 3.7;\ b = 0.07;\ c = 1.5;\ x = 5.75$

17. $y = 4.5\frac{(a+b)^2}{(a-b)^2} - \sqrt{(a+b)(a-b)} + 10^{-1}\frac{\ln(a-b)}{\ln(a+b)} \cdot e^{x^2}$

$a = 7.5;\ b = 1.2;\ x = 0.5$

18. $y = 2.4\left|\frac{x^2+b}{a}\right| + (a-b)\sin^2(a-b) + 10^{-2}(x-b)$

$a = 5.1;\ b = 0.7;\ x = -0.05$

19. $y = \frac{ax - \sqrt{b}}{5.7(x^2+b^2)} - \frac{|x+b| - a^2}{x^2}\operatorname{tg}^2 b$

$a = 0.1;\ b = 2.4;\ x = -0.3$

20. $y = \sqrt{\frac{c - dx^2}{x}} + \frac{\ln(x^2+c)}{0.7x + ad} - \frac{10^{-2}}{c - dx^3}$

$a = 4.5;\ c = 7.4;\ d = -2.1;\ x = 0.15$

ЛАБОРАТОРНАЯ РАБОТА № 3.
РАЗВЕТВЛЯЮЩИЕСЯ АЛГОРИТМЫ

***Цель лабораторной работы**:* научиться пользоваться элементами управления для организации переключений (RadioButton). Написать и отладить программу разветвляющегося алгоритма.

3.1. 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 > (больше), < (меньше), != (не равно), >= (больше или равно), <= (меньше или равно), == (равно). Описываются логические переменные следующим образом:

```
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 ложен только тогда, когда оба операнда ложны.

3.2. 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.

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` – операторы.

При выполнении полной формы оператора `if` сначала вычисляется выражение `B`, затем анализируется его результат: если `B` истинно, то выполняется оператор `S1`, а оператор `S2` пропускается; если `B` ложно, то выполняется оператор `S2`, а `S1` – пропускается. Таким образом, с помощью полной формы оператора `if` можно выбрать одно из двух альтернативных действий процесса обработки данных.

Для примера вычислим значение функции:

$$y(x) = \begin{cases} \sin(x), \text{ если } x \le a \\ \cos(x), \text{ если } a < x < b \\ \text{tg}(x), \text{ если } x \ge b \end{cases}$$

Указанное выражение может быть запрограммировано в виде

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

или с помощью оператора `else`:

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

Важное примечание! В С-подобных языках программирования, к которым относится и С#, операция сравнения представляется двумя знаками равенства, например:

```
if (a == b)
```

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

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

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

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

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

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

```
            MessageBox.Show("Вторник");
        else
            if (dayOfWeek == 3)
                MessageBox.Show("Среда");
            else
                if (dayOfWeek == 4)
                    MessageBox.Show("Четверг");
                else
                    if (dayOfWeek == 5)
                        MessageBox.Show("Пятница");
                    else
                        if (dayOfWeek == 6)
                            MessageBox.Show("Суббота");
                        else
                            if (dayOfWeek == 7)
                                MessageBox.Show("Воскресенье");
                            else
                                MessageBox.Show("Неверное значение!");
```

3.3. Кнопки-переключатели

При создании программ в Visual Studio для организации разветвлений часто используются элементы управления в виде кнопок-переключателей (RadioButton). Состояние такой кнопки (включено–выключено) визуально отражается на форме, а в программе можно узнать его с помощью свойства Checked: если кнопка включена, это свойство будет содержать True, в противном случае False. Если пользователь выбирает один из вариантов переключателя в группе, все остальные автоматически отключаются.

Группируются радиокнопки с помощью какого-либо контейнера – часто это бывает элемент GroupBox. Радиокнопки, размещенные в разных контейнерах, образуют независимые группы.

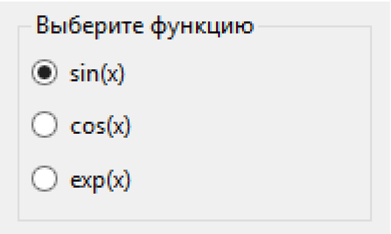


Рис. 3.1. Группа радиокнопок

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

3.4. Пример написания программы

З а д а н и е : ввести три числа – x, y, z. Вычислить

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

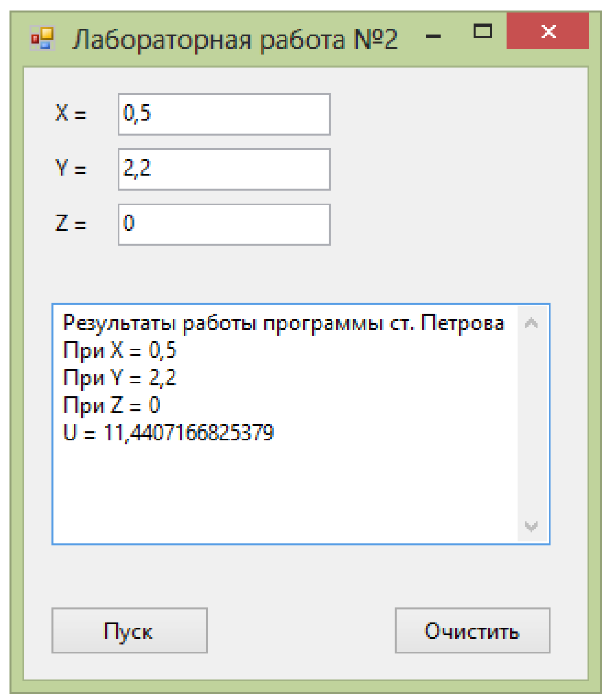


Рис. 3.2. Окно лабораторной работы

3.4.1. Создание формы

Создайте форму, в соответствии с рис. 3.2.

Разместите на форме элементы `Label`, `TextBox` и `Button`. Поле для вывода результатов также является элементом `TextBox` с установленным в `True` свойством `Multiline` и свойством `ScrollBars` установленным в `Both`.

3.4.2. Создание обработчиков событий

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(textBox2.Text);
    double y = Convert.ToDouble(textBox1.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 = " + textBox2.Text +
                         Environment.NewLine;
        textBox4.Text += "При Y = " + textBox1.Text +
                         Environment.NewLine;
        textBox4.Text += "При Z = " + textBox3.Text +
                         Environment.NewLine;
        // Вычисление выражения
        double u;
        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;
        // Вывод результата
        textBox4.Text += "U = " + u.ToString() +
           Environment.NewLine;
    }
```

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

Индивидуальные задания

По указанию преподавателя выберите индивидуальное задание из нижеприведенного списка. В качестве *f(x)* использовать по выбору: *sh(x)*, x^2, e^x. Отредактируйте вид формы и текст программы, в соответствии с полученным заданием.

Усложненный вариант задания для продвинутых студентов: с помощью радиокнопок (RadioButton) дать пользователю возможность во время работы программы выбрать одну из трех приведенных выше функций.

1. $$a = \begin{cases} (f(x)+y)^2 - \sqrt{f(x)y}, & xy\rangle 0 \\ (f(x)+y)^2 + \sqrt{|f(x)\,y|}, & xy\langle 0 \\ (f(x)+y)^2 + 1, & xy = 0. \end{cases}$$

2. $$b = \begin{cases} \ln(f(x)) + \left(f(x)^2 + y\right)^3, & x/y\rangle 0 \\ \ln|f(x)/y| + (f(x)+y)^3, & x/y\langle 0 \\ \left(f(x)^2 + y\right)^3, & x = 0 \\ 0, & y = 0. \end{cases}$$

3. $$c = \begin{cases} f(x)^2 + y^2 + \sin(y), & x - y = 0 \\ (f(x)-y)^2 + \cos(y), & x - y\rangle 0 \\ (y - f(x))^2 + \mathrm{tg}(y), & x - y\langle 0. \end{cases}$$

4. $$d = \begin{cases} (f(x)-y)^3 + \mathrm{arctg}(f(x)), & x\rangle y \\ (y - f(x))^3 + \mathrm{arctg}(f(x)), & y\rangle x \\ (y + f(x))^3 + 0.5, & y = x. \end{cases}$$

5. $e = \begin{cases} i\sqrt{f(x)}, & i - \textit{нечетное},\ x\rangle 0 \\ i/2\sqrt{|f(x)|}, & i - \textit{четное},\ x\langle 0 \\ \sqrt{|if(x)|}, & \textit{иначе.} \end{cases}$

6. $g = \begin{cases} e^{f(x)-|b|}, & 0.5\langle xb\langle 10 \\ \sqrt{|f(x)+b|}, & 0.1\langle xb\langle 0.5 \\ 2f(x)^2, & \textit{иначе.} \end{cases}$

7. $s = \begin{cases} e^{f(x)}, & 1\langle xb\langle 10 \\ \sqrt{|f(x)+4*b|}, & 12\langle xb\langle 40 \\ bf(x)^2, & \textit{иначе.} \end{cases}$

8. $j = \begin{cases} \sin\left(5f(x)+3m|f(x)|\right), & -1\langle m\langle x \\ \cos\left(3f(x)+5m|f(x)|\right), & x\rangle m \\ \left(f(x)+m\right)^2, & x = m. \end{cases}$

9. $l = \begin{cases} 2f(x)^3+3p^2, & x\rangle |p| \\ |f(x)-p|, & 3\langle x\langle |p| \\ \left(f(x)-p\right)^2, & x = |p|. \end{cases}$

10. $k = \begin{cases} \ln\left(|f(x)|+|q|\right), & |xq|\rangle 10 \\ e^{f(x)+q}, & |xq|\langle 10 \\ f(x)+q, & |xq| = 10 \end{cases}$

11. $m = \dfrac{\max\left(f(x), y, z\right)}{\min\left(f(x), y\right)} + 5.$

12. $n = \dfrac{\min\left(f(x)+y, y-z\right)}{\max\left(f(x), y\right)}.$

13. $p = \dfrac{\left|\min\left(f(x), y\right) - \max\left(y, z\right)\right|}{2}.$

14. $q = \dfrac{\max\left(f(x)+y+z, xyz\right)}{\min\left(f(x)+y+z, xyz\right)}.$

15. $c = \begin{cases} f(\sin(x))^2 + \sin\left(f(y)\right), & x-y=0 \\ \left(f(\cos(x))\right) + \cos\left(f(y)\right), & x-y\rangle 0 \\ \left(y - f(\mathrm{tg}(x))\right)^2 + \mathrm{tg}\left(y\right), & x-y\langle 0. \end{cases}$

16. $a = \begin{cases} \dfrac{\left(\mathrm{ax}^2+2\right)}{\left(\mathrm{x}^2+1\right)} f(x), & 1<|\mathrm{x}|<3, \\ a^2 + f(x), & |\mathrm{x}| \ \geq 3 \\ \mathrm{ax}\dfrac{\mathrm{f(x)}}{\left(\mathrm{x}+2\right)}, & |\mathrm{x}| \ \leq 1. \end{cases}$

17. $c = \begin{cases} f(x)^3 - y^3 \cdot \cos(x), & x+y=0 \\ \left(f(x)\cdot y\right)^2 - \cos(y), & x+y\rangle 0 \\ \left(y \cdot f(x)\right)^2 + \pi, & x+y\langle 0. \end{cases}$

18. $k = \begin{cases} \ln\left(|f(x^2)|+|k|\right), & |x\cdot k|\rangle 10 \\ \pi^{f(x)+q}, & |x\cdot k|\langle 10 \\ f(x)-k, & |x\cdot k| = 10 \end{cases}$

19. $c = \begin{cases} \sin(f(x)) + \cos(f(y)), & x-y=0 \\ \mathrm{tg}(f(x+y), & x-y\rangle 0 \\ \sin^2(f(x)) + \cos^2(f(y)), & x-y\langle 0. \end{cases}$

20. $r = \max\left(\min\left(f(x), y\right), z\right).$

ЛАБОРАТОРНАЯ РАБОТА № 4.
ЦИКЛИЧЕСКИЕ АЛГОРИТМЫ

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

4.1. 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.

4.2. Цикл с предусловием

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 зацикливания, 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 целых чисел от 1 до n по нажатию кнопки на форме:

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

```
    int n = 10;  // Количество повторений цикла
    int i = 1;    // Начальное значение
    while (i <= n) // Пока i меньше или равно n
    {
        MessageBox.Show(i.ToString()); // Показываем i
        i++; // Увеличиваем i на 1
    }
}
```

4.3. Цикл с постусловием

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 зацикливания в случае, если условие `B` всегда будет оставаться истинным.

4.4. Цикл с параметром

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 пара-

метром 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:

```
string s = ""; // Инициализация строки
for (int i = 0; i <= 9; i++) // Перечисление всех чисел
    s += i.ToString() + " "; // Добавляем число и пробел
MessageBox.Show(s.ToString()); // Показываем результат
```

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 – ложно и управление передается за пределы цикла.

4.5. 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.

Список ошибок

Ошибок: 2 | Предупреждений: 0 | Сообщений: 0

	Описание	Файл	Строка
1	Использование локальной переменной "u", которой не присвоено значение	Form1.cs	45
2	Аргумент отсутствует	Form1.cs	39

Рис. 4.1. Окно со списком ошибок компиляции

При выявлении ошибок компиляции в нижней части экрана появляется текстовое окно (см. 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 необходимо исправлять их последовательно, сверху вниз и, после исправления каждой – 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).

```
double y = Convert.ToDouble(textBox2.Text);
double z = Convert.ToDouble(textBox3.Text);
// Ввод исходных данных и их вывод в окно результатов
textBox4.Text = "Результаты работы программы ст. Петрова И.И. " + Er
textBox4.Text += "При X = " + textBox1.Text + Environment.NewLine;
textBox4.Text += "При Y = " + textBox2.Text + Environment.NewLine;
textBox4.Text += "При Z = " + textBox3.Text + Environment.NewLine;
int n = 0;
if (radioButton2.Checked) n = 1; else n = 2;
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;
```

Рис. 4.2. Фрагмент кода с точкой останова

При выполнении программы и достижения установленной точки программа будет остановлена, и далее можно выполнять код по шагам с помощью команд *Отладка → Шаг с обходом* (без захода в методы) или *Отладка → Шаг с заходом* (с заходом в методы) (рис. 4.3).

```
textBox4.Text += "При Z = " + textBox3.Text + Environment.NewLine;
int n = 0;
if (radioButton2.Checked) n = 1;
else if (radioButton3.Checked) n = 2;
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
```

Рис. 4.3. Отладка программы

Желтым цветом выделяется оператор, который будет выполнен. Значение переменных во время выполнения можно увидеть, наведя на них курсор. Для прекращения отладки и остановки программы нужно выполнить команду меню *Отладка* → *Остановить отладку*.

Для поиска алгоритмических ошибок можно контролировать значения промежуточных переменных на каждом шаге выполнения подозрительного кода и сравнивать их с результатами, полученными вручную.

4.6. Порядок выполнения задания

З а д а н и е: Вычислить и вывести 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. 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 следует проверить правильность работы программы с помощью контрольного примера (см. рис. 4.4). Установите точку останова на оператор перед циклом и запустите программу. После попадания на точку остановки, выполните пошагово программу и проследите, как меняются все переменные в процессе выполнения.

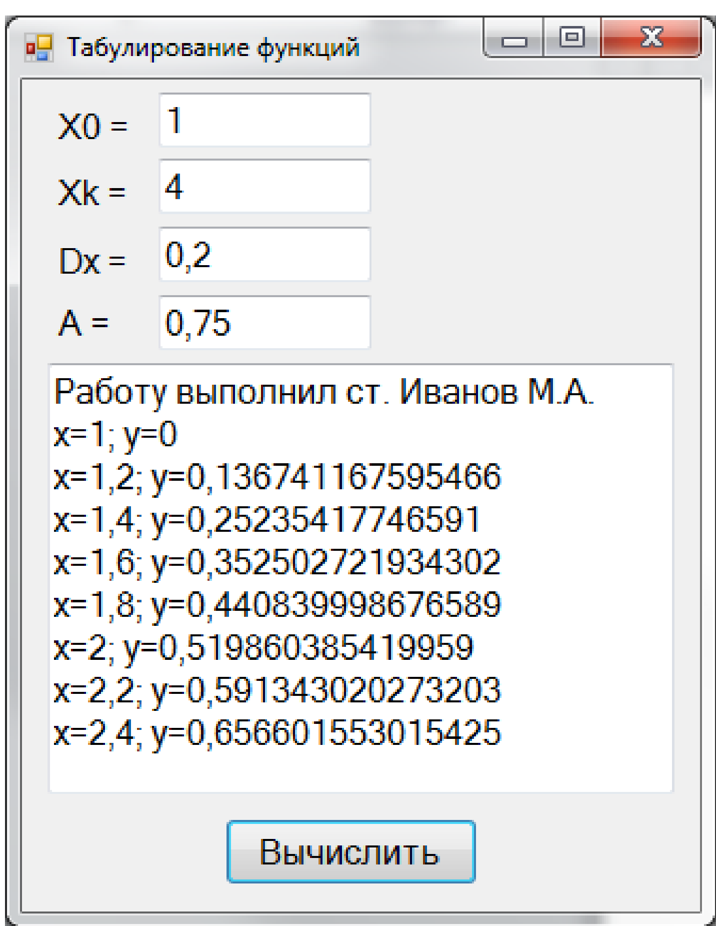


Рис. 4.4. Окно программы для табулирования функции

Индивидуальные задания

Составьте программу табулирования функции $y(x)$, выведите на экран значения x и $y(x)$. Нужный вариант задания выберите из нижеприведенного списка по указанию преподавателя. Откорректируйте элементы управления в форме в соответствии со своим вариантом задания.

1) $\mathrm{y} = 10^{-2} bc / x + \cos\sqrt{a^3 x},$
$x_0 = -1.5; x_k = 3.5; dx = 0.5;$
$a = -1.25; b = -1.5; c = 0.75;$

2) $\mathrm{y} = 1.2(\mathrm{a}-\mathrm{b})^3 e^{x^2} + x,$
$x_0 = -0.75; x_k = -1.5; dx = -0.05;$
$a = 1.5; b = 1.2;$

3) $y = 10^{-1} a x^3 tg(a - bx),$
$x_0 = -0.5; x_k = 2.5; dx = 0.05;$
$a = 10.2; b = 1.25;$

4) $y = a x^3 + \cos^2(x^3 - b),$
$x_0 = 5.3; x_k = 10.3; dx = 0.25;$
$a = 1.35; b = -6.25;$

5) $y = x^4 + \cos(2 + x^3 - d),$
$x_0 = 4.6; x_k = 5.8; dx = 0.2;$
$d = 1.3;$

6) $y = x^2 + tg(5x + b / x),$
$x_0 = -1.5; x_k = -2.5; dx = -0.5;$
$b = -0.8;$

7) $y = 9(x + 15\sqrt{x^3 + b^3}),$
$x_0 = -2.4; x_k = 1; dx = 0.2;$
$b = 2.5;$

8) $y = 9x^4 + \sin(57.2 + x),$
$x_0 = -0.75; x_k = -2.05; dx = -0.2;$

9) $y = 0.0025 b x^3 + \sqrt{x + e^{0.82}},$
$x_0 = -1; x_k = 4; dx = 0.5;$
$b = 2.3;$

10) $y = x \cdot \sin(\sqrt{x + b - 0.0084}),$
$x_0 = -2.05; x_k = -3.05; dx = -0.2;$
$b = 3.4;$

11) $y = x + \sqrt{|x^3 + a - b e^x|},$
$x_0 = -4; x_k = -6.2; dx = -0.2;$
$a = 0.1;$

12) $y = 9(x^3 + b^3) tg x,$
$x_0 = 1; x_k = 2.2; dx = 0.2;$
$b = 3.2;$

13) $y = |x - b|^{1/2} / |b^3 - x^3|^{3/2} + \ln|x - b|,$
$x_0 = -0.73; x_k = -1.73; dx = -0.1;$
$b = -2;$

14) $y = (x^{5/2} - b)\ln(x^2 + 12.7),$
$x_0 = 0.25; x_k = 5.2; dx = 0.3;$
$b = 0.8;$

15) $y = 10^{-3}|x|^{5/2} + \ln|x + b|,$
$x_0 = 1.75; x_k = -2.5; dx = -0.25;$
$b = 35.4;$

16) $y = 15.28|x|^{-3/2} + \cos(\ln|x| + b),$
$x_0 = 1.23; x_k = -2.4; dx = -0.3;$
$b = 12.6;$

17) $y = 0.00084(\ln|x|^{5/4} + b)/(x^2 + 3.82),$
$x_0 = -2.35; x_k = -2; dx = 0.05;$
$b = 74.2;$

18) $y = 0.8 \cdot 10^{-5}(x^3 + b^3)^{7/6},$
$x_0 = -0.05; x_k = 0.15; dx = 0.01;$
$b = 6.74;$

19) $y = (\ln(\sin(x^3 + 0.0025)))^{3/2} + 0.8 \cdot 10^{-3},$
$x_0 = 0.12; x_k = 0.64; dx = 0.2;$

20) $y = a + x^{2/3}\cos(x + e^x),$
$x_0 = 5.62; x_k = 15.62; dx = 0.5;$
$a = 0.41$

ЛАБОРАТОРНАЯ РАБОТА № 5. КЛАССЫ И ОБЪЕКТЫ

Цель лабораторной работы: изучить основные понятия, относящиеся к классам и объектам, освоить динамическое создание объектов в программном коде.

5.1. Классы и объекты

В объектно-ориентированном подходе существуют понятия *класс* и *объект*.

Класс – это программная единица, которая задает общий шаблон для конкретных объектов. Класс содержит все необходимые описания переменных, свойств и методов, которые относятся к объекту. Примером класса в реальной жизни является *понятие «автомобиль»*: как правило, автомобиль содержит некоторое количество колес, дверей, имеет какой-то цвет, но эти конкретные детали в классе не описываются.

Объект – это экземпляр класса. Свойства объекта содержат конкретные данные, характерные для данного экземпляра. В реальной жизни примером объекта будет конкретный экземпляр автомобиля с 4 колесами, 5 дверками и синего цвета.

5.2. Динамическое создание объектов

Чаще всего для размещения на форме кнопки, поля ввода или других управляющих элементов используется дизайнер среды Visual Studio: нужный элемент выделяется в панели элементов и размещается на форме. Однако иногда создавать элементы нужно уже в процессе выполнения программы. Поскольку каждый элемент управления представляет собой отдельный класс, его помещение на форму программным способом включает несколько шагов:

1. Создание экземпляра класса.
2. Привязка его к форме.
3. Настройка местоположения, размеров, текста и т. п.

Например, чтобы создать кнопку, нужно выполнить следующий код (его следует разместить в обработчике сообщения `Load` или в какомлибо другом методе):

```
Button b = new Button();
```

Здесь объявляется переменная b, относящаяся к классу Button, как и в предыдущих лабораторных работах. Однако дальше идет нечто новое: с помощью оператора new создается экземпляр класса Button, и ссылка на него присваивается переменной b. При этом выполняется целый ряд дополнительных действий: выделяется память под объект, инициализируются все свойства и переменные.

Далее нужно добавить объект на форму. Для этого служит свойство Parent, которое определяет родительский элемент, на котором будет размещена кнопка:

```
b.Parent = this;
```

Ключевое слово this относится к тому объекту, в котором размещен выполняемый в данный момент метод. Поскольку все методы в лабораторных работах размещаются в классе формы, то и this относится к этому конкретному экземпляру формы.

Вместо формы кнопку можно поместить на другой контейнер. Например, если на форме есть элемент управления Panel, то можно поместить кнопку на него следующим образом:

```
b.Parent = panel1;
```

Чтобы задать положение и размеры кнопки, нужно использовать свойства Location и Size:

```
b.Location = new Point(10, 20);
b.Size = new Size(200, 100);
```

Обратите внимание, что Location и Size – это тоже объекты. Хотя внутри у Location содержатся координаты x и y, задающие левый верхний угол объекта, не получится поменять одну из координат, нужно менять целиком весь объект Location. То же самое относится и к свойству Size.

На самом деле, каждый раз, когда на форму помещается новый элемент управления или вносятся какие-то изменения в свойства элементов управления, Visual Studio генерирует специальный служебный код, который проделывает приведенные выше операции по созданию и настройке элементов управления. Попробуйте поместить на форму кнопку, изменить у нее какие-нибудь свойства, а затем найдите в обозревателе решений ветку формы Form1, разверните ее и сделайте двойной щелчок по ветке Form1.Designer.cs. Откроется файл с текстом программы на языке C#, которую среда создала автоматически. Менять этот код вручную крайне не рекомендуется! Однако можно его изучить, чтобы понять принципы создания элементов управления в ходе выполнения программы.

5.3. Область видимости

Переменные, объявленные в программе, имеют область видимости. Это значит, что переменная, описанная в одной части программы, не обязательно будет видна в другой. Вот наиболее часто встречающиеся ситуации:

1. Переменные, описанные внутри метода, не будут видны за пределами этого метода. Например:

```
void MethodA()
{
    // Описываем переменную delta
    int delta = 7;
}

void MethodB()
{
    // Ошибка: переменная delta в этом методе неизвестна!
    int gamma = delta + 1;
}
```

2. Переменные, описанные внутри блока или составного оператора, видны только внутри этого блока. Например:

```
void Method()
{
    if (a == 7)
    {
        int b = a + 5;
    }
    // Ошибка: переменная b здесь уже неизвестна!
    MessageBox.Show(b.ToString());
}
```

3. Переменные, описанные внутри класса, являются *глобальными* и доступны для всех методов этого класса, например:

```
class Form1 : Form
{
    int a = 5;

    void Method()
```

```
        {
            // Переменная a здесь действительна
            MessageBox.Show(a.ToString());
        }
    }
```

5.4. Операции `is` и `as`

Часто бывает удобно переменные разных классов записать в один список, чтобы было легче его обрабатывать. Чтобы проверить, к какому классу принадлежит какой-либо объект, можно использовать оператор `is`: он возвращает истину, если объект принадлежит указанному классу. Пример:

```
Button b = new Button();
if (b is Button)
   MessageBox.Show("Это кнопка!");
else
   MessageBox.Show("Это что-то другое...");
```

Как правило, в общих списках объекты хранятся в «обезличенном» состоянии, так, чтобы у всех у них был лишь минимальный общий для всех набор методов и свойств. Для того чтобы получить доступ к расширенным свойствам объекта, нужно *привести* его к исходному классу с помощью *операции приведения* `as`:

```
(someObject as Button).Text = "Это кнопка!";
```

Следует помнить, что операция приведения сработает только в том случае, если объект изначально принадлежит тому классу, к которому его пытаются привести (или совместим с ним), в противном случае оператор `as` выбросит исключение и остановит выполнение программы. Поэтому более безопасный подход состоит в комбинированном применении операторов `as` и `is`: сначала проверяем совместимость объекта и класса, и только потом выполняем операцию приведения:

```
if (someObject is Button)
   (someObject as Button).Text = "Это кнопка!";
```

В качестве практического примера использования этих операций рассмотрим пример программы, которая перебирает все элементы управления на форме, и у кнопок (но не у других элементов управления!) заменяет текст на пять звездочек «*****»:

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

```
        // Перебираем все элементы управления
        foreach (Control c in this.Controls)
            if (c is Button) // Кнопка?
                (c as Button).Text = "*****"; // Да!
    }
```

5.5. Сведения, передаваемые в событие

Когда происходит какое-либо событие (например, событие `Click` при нажатии на кнопку), в обработчик этого события передаются дополнительные сведения об этом событии в параметре `e`.

Например, при щелчке кнопки мыши на объекте возникает событие `MouseClick`. Для этого события параметр `e` содержит целый ряд переменных, которые позволяют узнать информацию о нажатии:

- `Button` – какая кнопка была нажата;
- `Clicks` – сколько раз была нажата и отпущена кнопка мыши;
- `Location` – координаты точки, на которую указывал курсор в момент нажатия, в виде объекта класса `Point`;
- `X` и `Y` – те же координаты в виде отдельных переменных.

Индивидуальные задания

Если в индивидуальном задании используется элемент `Panel`, измените его цвет, чтобы он визуально выделялся на форме. Если используется элемент `Label`, не забудьте присвоить ему какой-либо текст, иначе он не будет виден на форме.

1. Разработать программу, динамически порождающую на окне кнопки. Левый верхний угол кнопки определяется местоположением курсора при щелчке. Вывести надпись на кнопке с координатами ее левого верхнего угла.
2. Разработать программу, динамически порождающую на окне кнопки и поля ввода. Левый верхний угол элемента управления определяется местоположением курсора при щелчке. Кнопка порождается, если курсор находится в левой половине окна, в ином случае порождается поле ввода.
3. На форме размещен элемент управления `Panel`. Написать программу, которая при щелчке мыши на элементе управления Panel добавляет в него кнопки `Button`, а при щелчке на форме в нее добавляются поля ввода `TextBox`.
4. На форме размещены 3 панели (элемент управления Panel). Написать программу, которая при щелчке мыши на первой панели добавляет во вторую панель кнопки Button, при щелчке на второй панели добавляет в третью панель поля ввода TextBox, а при щелчке на третьей панели добавляет на первую панель метки Label.

5. Написать программу, добавляющую на форму кнопки. Кнопки добавляются в узлы прямоугольной сетки. Расстояния между кнопками и расстояния между крайней кнопкой и границей окна должны быть равны как по горизонтали, так и по вертикали.

6. Разработать программу, при щелчке мыши динамически порождающую на окне кнопки или поля ввода. Каждый четный элемент управления является кнопкой, нечетный – полем ввода. Левый верхний угол кнопки определяется местоположением курсора при щелчке. Для поля ввода положение курсора определяет координаты *правого нижнего* угла.

7. Создать программу с кнопкой, меткой и полем ввода. При щелчке на соответствующий элемент на форме динамически должен создаваться подобный ему элемент. Предусмотреть возможность вывода количества кнопок, меток и полей ввода.

8. Создать программу, добавляющую различные элементы управления на форму и на панель `Panel`. Тип элементов управления выбирается случайным образом. Предусмотреть возможность вывода информации о количестве элементов по типам и информацию о расположении элементов.

9. Разработать программу, добавляющую на форму последовательность элементов управления случайной длины. Тип элементов управления задается случайным образом. Предусмотреть возможность вывода информации о количестве элементов по типам.

10. Написать программу, динамически порождающую на окне кнопки или метки. Левый верхний угол элемента управления определяется местоположением курсора при щелчке. При нажатии правой кнопки мыши на форме с нее удаляются все кнопки.

11. Написать программу, динамически порождающую на окне поочередно кнопки или поля ввода. Левый верхний угол элемента управления определяется местоположением курсора при щелчке. При нажатии правой кнопки мыши на форме с нее удаляются все порожденные элементы.

12. Разработать программу с двумя кнопками на форме. При нажатии на первую на форму добавляется одна панель `Panel`. При нажатии на вторую кнопку в каждую панель добавляется поле ввода.

13. Разработать программу с двумя кнопками на форме. При нажатии на первую на форму добавляется одна кнопка или поле ввода. При нажатии на вторую кнопку каждое поле увеличивается по вертикали в два раза.

14. Написать программу с кнопкой и тремя полями ввода. При нажатии на кнопку программа анализирует содержимое первого поля и динамически порождает элемент управления. Если в первом поле вво-

да содержится буква «К», то на форму добавляется кнопка, если «П» – поле ввода, если «М» – метка. Во втором и третьем поле ввода содержатся координаты левого верхнего угла будущего элемента управления.

15. Разработать программу, добавляющую на форму метки с текстом. Местоположение и размеры меток определяются в программе динамически через поля ввода. В заголовок окна, анализируя размер всех меток, вывести количество маленьких и больших меток. Маленькой меткой считается метка размером менее 50 пикселей по горизонтали и вертикали.

16. Создать программу с двумя кнопками на форме, динамически порождающую на окне метки или поля ввода. При нажатии на первую кнопку каждая метка увеличивается по горизонтали в два раза. При нажатии на вторую кнопку каждое поле уменьшается по вертикали в два раза.

17. Разработать программу, динамически порождающую на окне кнопки и поля ввода. Координаты элемента управления определяются случайным образом. Элементы управления не должны накладываться друг на друга. Если нет возможности добавить элемент управления (нет места для размещения элемента), то предусмотреть вывод информации об этом.

18. Разработать программу, динамически порождающую на окне кнопки и поля ввода. Координаты элемента управления определяются случайным образом. При наведении курсора на элемент управления он должен быть удален с формы.

19. Разработать программу, динамически порождающую при щелчке на окне различные элементы (поля ввода, кнопки, метки). Тип элементов определяется с помощью радиокнопок. Все элементы располагаются горизонтально в ряд. При достижении правой границы окна начинается новый ряд элементов.

20. Разработать программу, динамически порождающую или поле ввода (при нажатии на окне левой кнопкой мыши), или кнопку (при нажатии на окне правой кнопкой мыши). Все элементы располагаются наискосок, начиная с левого верхнего угла окна. Реализовать обработчик события изменения размера окна, в котором удалить все порожденные элементы.

ЛАБОРАТОРНАЯ РАБОТА № 6.
СТРОКИ

Цель лабораторной работы: изучить правила работы с элементом управления ListBox. Написать программу для работы со строками.

6.1. Строковый тип данных

Для хранения строк в языке C# используется тип string. 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 (счет 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:

- \" – Кавычка.
- \\ – Обратная косая черта.
- \n – Новая строка.
- \r – Возврат каретки.
- \t – Горизонтальная табуляция.

6.2. Более эффективная работа со строками

Строки типа string представляют собой неизменяемые объекты: после того, как строка инициализирована, изменить ее уже нельзя. Рассмотрим для примера следующий код:

```
string s = "Hello, ";
s += "world!";
```

Здесь компилятор создает в памяти строковый объект и инициализирует его строкой «*Hello,* », а затем создает другой строковый объект и инициализирует его значением первого объекта и новой строкой «*world!*», а затем заменяет значение переменной `s` на новый объект. В результате строка `s` содержит именно то, что хотел программист, однако в памяти остается и изначальный объект со строкой «*Hello,* ». Конечно, со временем сборщик мусора уничтожит этот бесхозный объект, однако если в программе идет интенсивная работа со строками, то таких бесхозных объектов может оказаться очень много. Как правило, это негативно сказывается на производительности программы и объеме потребляемой ею памяти.

Чтобы компилятор не создавал каждый раз новый строковый объект, разработчики языка C# ввели другой строковый класс: `StringBuilder`. Приведенный выше пример с использованием этого класса будет выглядеть следующим образом:

```
StringBuilder s = new StringBuilder("Hello, ");
s.Append("world!");
```

Конечно, визуально этот код выглядит более сложным, зато при активном использовании строк в программе он будет гораздо эффективнее. Помимо добавления строки к существующему объекту (`Append`) класс `StringBuilder` имеет еще ряд полезных методов:

- `Insert`: вставляет указанный текст в нужную позицию исходной строки
- `Remove`: удаляет часть строки
- `Replace`: заменяет указанный текст в строке на другой.

Если нужно преобразовать объект `StringBuilder` в обычную строку, то для этого можно использовать метод `ToString()`:

```
StringBuilder s = new StringBuilder("Яблоко");
string a = s.ToString();
```

6.3. Элемент управления `ListBox`

Элемент управления `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, *приведенные* 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`.

6.4. 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:

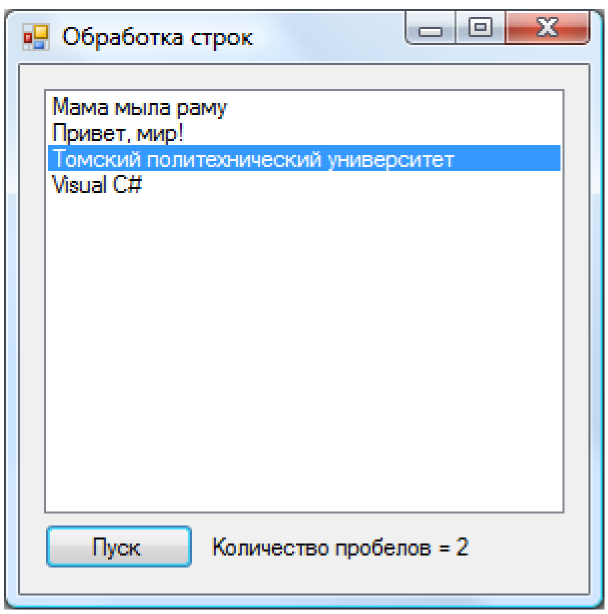


Рис. 6.1. Окно программы обработки строк

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)
        {
            // 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`.

1. Дана строка, состоящая из групп нулей и единиц. Посчитать количество нулей и единиц.
2. Посчитать в строке количество слов.
3. Найти количество знаков препинания в исходной строке.
4. Дана строка символов. Вывести на экран цифры, содержащиеся в строке.
5. Дана строка символов. Сформировать новую строку, в которую включить все символы исходной строки, стоящие на четных местах. При этом должен быть обратный порядок следования символов по отношения к исходной строке.
6. Сформировать и вывести две новых строки на основе входной строки символов. В первую строку включить все символы, стоящие на четных местах, во вторую – символы, стоящие на нечетных местах в исходной строке.
7. Дана строка символов, состоящая из произвольных десятичных цифр, разделенных пробелами. Вывести количество четных чисел в этой строке.
8. Дана строка символов. Вывести на экран количество строчных русских букв, входящих в эту строку.

9. Сформировать и вывести три новых строки на основе входной строки символов. В первую строку включить все цифры, во вторую – все знаки препинания: точки, запятые, двоеточия, точки с запятой, восклицательные и вопросительные знаки, в третью строку – все остальные символы. Например, входная строка содержит: «выходные дни: 1, 2 января, 8 марта, 1 мая, 9 мая!», после обработки первая строка будет содержать: «12819», вторая строка: «:,,,!», третья строка: «выходные дни января марта мая мая».

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

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

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

13. 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.

14. Дана строка символов, состоящая из произвольного текста на английском языке, слова разделены пробелами. Заменить все буквы латинского алфавита на знак «+».

15. Дана строка символов, содержащая некоторый текст на русском языке. Заменить все большие буквы «А» на символ «*».

16. 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»).

17. Дана строка символов, состоящая из произвольного текста на английском языке, слова разделены пробелами. Сформировать новую строку, состоящую из чисел длин слов в исходной строке.

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

19. Поменять местами первое и второе слово в исходной строке.

20. Сформировать новую строку, где поменять местами первое и последнее слово из исходной строки.

ЛАБОРАТОРНАЯ РАБОТА № 7.
ОДНОМЕРНЫЕ МАССИВЫ

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

7.1. 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, определяет размер массива.

В общем случае процесс объявления переменной типа массив и выделение необходимого объема памяти может быть разделен. Кроме того, на этапе объявления массива можно произвести его инициализацию. Поэтому для объявления одномерного массива может использоваться одна из следующих форм записи:

```
тип[] имя_массива;
```

В этом случае описывается ссылка на одномерный массив, которая в дальнейшем может быть использована для адресации на уже существующий массив. Размер массива при таком объявлении не задается. Пример, в котором объявляется массив целых чисел с именем `a`:

```
int[] a;
```

Другая форма объявления массива включает и его инициализацию указанным количеством элементов:

```
тип[] имя_массива = new тип[размер];
```

В этом случае объявляется одномерный массив указанного типа и выделяется память под указанное количество элементов. Адрес данной области памяти записывается в ссылочную переменную. Элементы массива инициализируются значениями, которые по умолчанию приняты для данного типа: массивы числовых типов инициализируются нулями, строковые переменные – пустыми строками, символы – пробелами, объекты ссылочных типов – значением `null`. Пример такого объявления:

```
int[] a = new int[10];
```

Здесь выделяется память под 10 элементов типа `int`.

Наконец, третья форма записи дает возможность сразу инициализировать массив конкретными значениями:

```
тип[] имя_массива = {список инициализации};
```

При такой записи выделяется 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. Пример:

```
int[] a = { 0, 1, 2, 3 };
```

В данном случае будет создан массив `a`, состоящий из четырех элементов, и каждый элемент будет инициализирован очередным значением из списка.

Обращение 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]`. Следует помнить, что нумерация элементов начинается с нуля!

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 };
for (int i = 0; i < 10; i++)
    MessageBox.Show(myArray[i]);
```

7.2. Случайные числа

Одним из способов инициализации массива является задание элементов через случайные числа. Для работы со случайными числами используют класс `Random`. Метод `Random.Next` создает случайное число в диапазоне значений от нуля до максимального значения типа `int` (его можно узнать с помощью свойства `Int32.MaxValue`). Для создания случайного числа в диапазоне от нуля до какого-либо другого положительного числа используется перегрузка метода `Random.Next(Int32)` – единственный параметр метода указывает верхнюю границу диапазона, сама граница в диапазон не включается. Для создания случайного числа в другом диапазоне используется перегрузка метода `Random.Next(Int32, Int32)` – первый аргумент задает нижнюю границу диапазона, а второй – верхнюю.

7.3. Порядок выполнения индивидуального задания

Создайте форму с элементами управления, как показано на рис. 7.1. 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.

```
// Глобальная переменная видна всем методам
int[] Mas = new int[15];

// Заполнение исходного массива
private void button1_Click(object sender, EventArgs e)
{
    // Очищаем элемент управления
    listBox1.Items.Clear();
    // Инициализируем класс случайных чисел
    Random rand = new Random();
    // Генерируем и выводим 15 элементов
    for (int i = 0; i < 15; i++)
    {
        Mas[i] = rand.Next(-50, 50);
```

```
            listBox1.Items.Add("Mas[" + i.ToString() +
                "] = " + Mas[i].ToString());
        }
    }

    // Замена отрицательных элементов нулями
    private void button2_Click(object sender, EventArgs e)
    {
        // Очищаем элемент управления
        listBox2.Items.Clear();
        // Обрабатываем все элементы
        for (int i = 0; i < 15; i++)
        {
            if (Mas[i] < 0)
                Mas[i] = 0;
            listBox2.Items.Add("Mas[" + Convert.ToString(i)
                + "] = " + Mas[i].ToString());
        }
    }
```

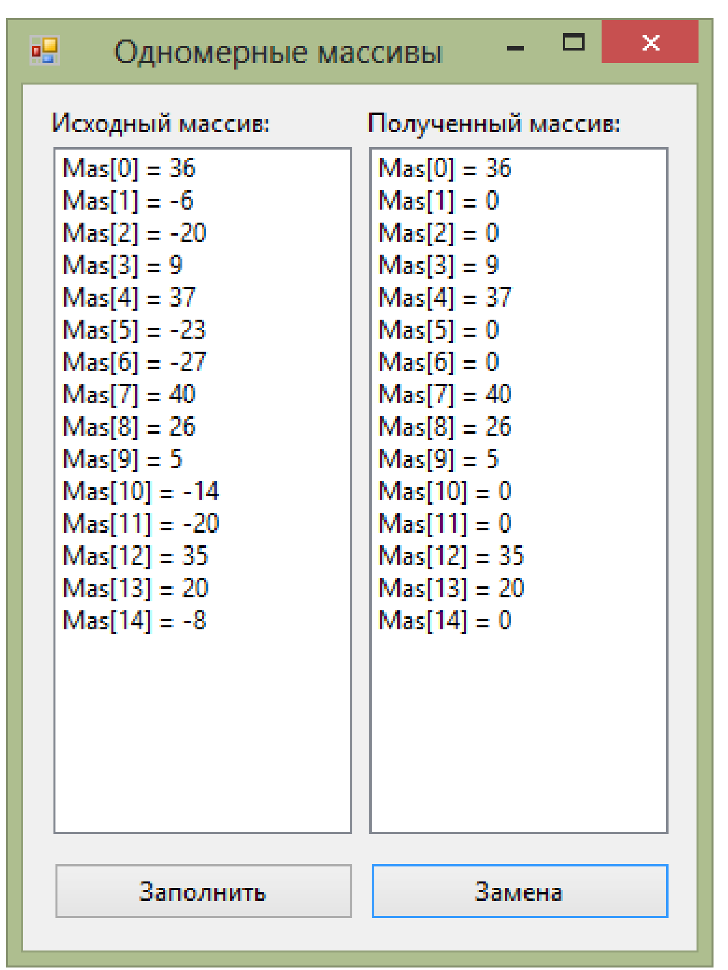


Рис. 7.1. Окно программы для работы с одномерными массивами

Индивидуальные задания

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

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

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

4. 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.

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

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

7. 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.

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

9. 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.

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

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

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

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

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

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

16. Массив задан датчиком случайных чисел на интервале [–33, 66]. Найти наименьший нечетный элемент.

17. Разработать программу, выводящую количество максимальных элементов в массиве из пятидесяти целочисленных элементов.

18. Разработать программу, циклически сдвигающую элементы целочисленного массива влево. Нулевой элемент массива ставится на последнее место, остальные элементы сдвигаются влево на одну позицию. Запрещается использовать второй массив.

19. Дано два массива с неубывающими целыми числами. Напишите программу, формирующую новый массив из элементов первых двух. В результирующем массиве не должно быть одинаковых элементов.

20. Дан массив целых чисел из 30 элементов. Найдите все локальные максимумы. (Элемент является локальным максимумом, если он не имеет соседей, больших, чем он сам).

ЛАБОРАТОРНАЯ РАБОТА № 8. МНОГОМЕРНЫЕ МАССИВЫ

Цель лабораторной работы: изучить свойства элемента управления `DataGridView`. Написать программу с использованием двухмерных массивов.

8.1. Двухмерные массивы

Многомерные массивы имеют более одного измерения. Чаще всего используются двумерные массивы, которые представляют собой таблицы. Каждый элемент такого 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}};
  ```

В качестве примера рассмотрим код, который строит «таблицу умножения» – каждая ячейка будет содержать значение, равное произведению номера строки и номера столбца:

```
// Объявление двухмерного массива
int[,] mul = new int[10,10];
// Заполнение массива
for (int i = 0; i < 10; i++)
  for (int j = 0; j < 10; j++)
    mul[i, j] = i * j;
```

8.2. Элемент управления `DataGridView`

При работе с двухмерными 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 = "*";
```

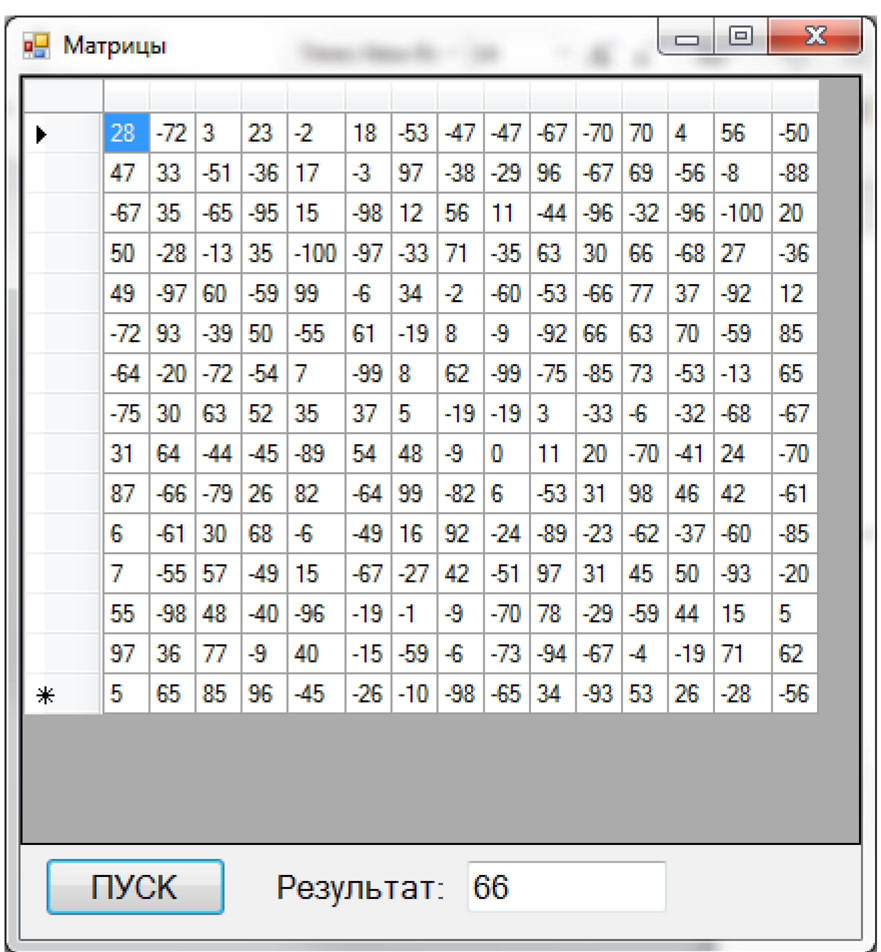
Этот код запишет во вторую строку и седьмой столбец знак звездочки.

8.3. Порядок выполнения задания

В ходе выполнения задания нужно создать программу для определения целочисленной матрицы 15×15. Разработать обработчик кнопки, который будет искать минимальный элемент на дополнительной диагонали матрицы. Результат вывести в текстовое поле.

Окно программы приведено на рис. 8.1.

Текст обработчика события нажатия на кнопку следует ниже.



28	-72	3	23	-2	18	-53	-47	-47	-67	-70	70	4	56	-50
47	33	-51	-36	17	-3	97	-38	-29	96	-67	69	-56	-8	-88
-67	35	-65	-95	15	-98	12	56	11	-44	-96	-32	-96	-100	20
50	-28	-13	35	-100	-97	-33	71	-35	63	30	66	-68	27	-36
49	-97	60	-59	99	-6	34	-2	-60	-53	-66	77	37	-92	12
-72	93	-39	50	-55	61	-19	8	-9	-92	66	63	70	-59	85
-64	-20	-72	-54	7	-99	8	62	-99	-75	-85	73	-53	-13	65
-75	30	63	52	35	37	5	-19	-19	3	-33	-6	-32	-68	-67
31	64	-44	-45	-89	54	48	-9	0	11	20	-70	-41	24	-70
87	-66	-79	26	82	-64	99	-82	6	-53	31	98	46	42	-61
6	-61	30	68	-6	-49	16	92	-24	-89	-23	-62	-37	-60	-85
7	-55	57	-49	15	-67	-27	42	-51	97	31	45	50	-93	-20
55	-98	48	-40	-96	-19	-1	-9	-70	78	-29	-59	44	15	5
97	36	77	-9	40	-15	-59	-6	-73	-94	-67	-4	-19	71	62
5	65	85	96	-45	-26	-10	-98	-65	34	-93	53	26	-28	-56

Рис. 8.1. Окно программы для работы с двухмерным массивом

```
private void button1_Click(object sender, EventArgs e)
{
    dataGridView1.RowCount = 15;  // Кол-во строк
```

```
        dataGridView1.ColumnCount = 15; // Кол-во столбцов
        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 =
                    a[i, j].ToString();
        // 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

1. Дана матрица A(3,4). Найти наименьший элемент в каждой строке матрицы. Вывести исходную матрицу и результаты вычислений.
2. Дана матрица A(3,3). Вычислить сумму второй строки и произведение первого столбца. Вывести исходную матрицу и результаты вычислений.
3. 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 $S \le 10$, то $b_{ij} = b_{ij}^2 - 1.5$. Вывести сумму S и преобразованную матрицу.
4. Дана матрица F(15,15). Вывести номер и среднее арифметическое элементов строки, начинающейся с 1. Если такой строки нет, то вывести сообщение «*Строки нет*».
5. Дана матрица F(7,7). Найти наименьший элемент в каждом столбце. Вывести матрицу и найденные элементы.
6. Найти наибольший элемент главной диагонали матрицы A(15,15) и вывести всю строку, в которой он находится.
7. Найти наибольшие элементы каждой строки матрицы Z(16,16) и поместить их на главную диагональ. Вывести полученную матрицу.
8. 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.

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

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

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

12. Найти номер столбца матрицы, в котором находится наименьшее количество положительных элементов.

13. Дан двухмерный массив 20×20 целочисленных элементов. Найдите все локальные максимумы. (Элемент является локальным максимумом, если он не имеет соседей, больших, чем он сам).

14. Дана матрица 7×7. Найти наибольший элемент среди стоящих на главной и побочной диагоналях и поменять его местами с элементом, стоящим на пересечении этих диагоналей.

15. Задана матрица, содержащая N строк и M столбцов. Седловой точкой этой матрицы назовем элемент, который одновременно является минимумом в своей строке и максимумом в своем столбце. Найдите количество седловых точек заданной матрицы.

16. Дана квадратная матрица 10×10. Реализуйте программу для транспонирования матрицы по главной и побочной диагоналям.

17. Требуется совершить обход квадратной матрицы по спирали так, как показано на рисунке: заполнение происходит с единицы из левого верхнего угла и заканчивается в центре числом N^2, где N – порядок матрицы. Реализуйте программу для матрицы 10×10.

1	2	3	4	5
16	17	18	19	6
15	24	25	20	7
14	23	22	21	8
13	12	11	10	9

18. Требуется заполнить змейкой квадратную матрицу так, как показано на рисунке: заполнение происходит с единицы из левого верхнего угла и заканчивается в правом нижнем числом N^2, где N – порядок матрицы. Реализуйте программу для матрицы 10×10.

1	3	4	10
2	5	9	11
6	8	12	15
7	13	14	16

19. Дана шахматная доска (матрица 8×8). Разработать программу, показывающую последовательность ходов конем с произвольной клетки. Конь ходит в соответствии с шахматными правилами, но в произвольную сторону (сгенерировать случайным образом). В клетку, с которой начинается ход, выводится единица. В клетку, в которую идет далее конь, записывается двойка и т. д. Ходить конем на клетки, на которых уже побывал конь, нельзя. Алгоритм останавливает работу, когда конем ходить некуда. Максимальная последовательность ходов – 64.

20. Проверка на симпатичность. Рассмотрим таблицу, содержащую n строк и m столбцов, в каждой клетке которой расположен ноль или единица. Назовем такую таблицу симпатичной, если в ней нет ни одного квадрата 2 на 2, заполненного целиком нулями или целиком единицами. Так, например, таблица 4 на 4, расположенная слева, является симпатичной, а расположенная справа таблица 3 на 3 – не является.

1	0	1	0
1	1	1	0
0	1	0	1
0	0	0	0

0	0	1
0	0	1
1	1	1

ЛАБОРАТОРНАЯ РАБОТА № 9.
ГРАФИКИ ФУНКЦИЙ

Цель лабораторной работы: изучить возможности построения графиков с помощью элемента управления `Chart`. Написать и отладить программу построения на экране графика заданной функции.

9.1. Как строится график с помощью элемента управления `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. 9.1).

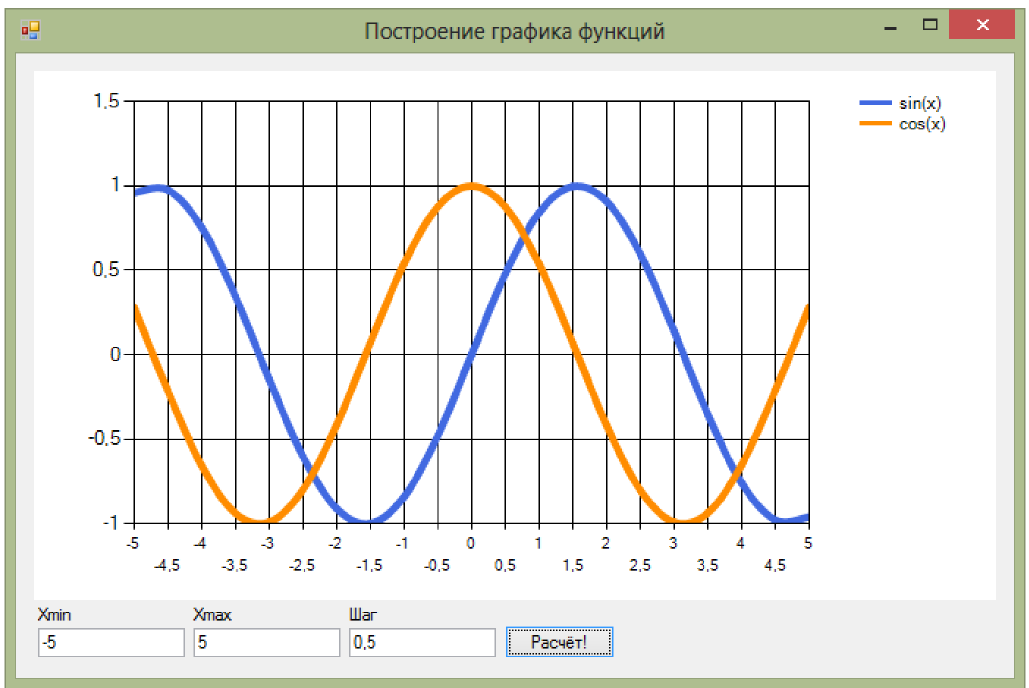


Рис. 9.1. Окно программы с элементом управления

Построение графика (диаграммы) производится после вычисления таблицы значений функции y = f(x) на интервале [Xmin, Xmax] с заданным шагом. Полученная таблица передается в специальный массив `Points` объекта `Series` элемента управления `Chart` с помощью метода `DataBindXY`. Элемент управления `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. В элементе

управления `Chart` можно настроить толщину, стиль и цвет линий, параметры шрифта подписей, шаги разметки координатной сетки и многое другое. В процессе работы программы изменение параметров возможно через обращение к соответствующим свойствам элемента управления `Chart`. Так, например, свойство `AxisX` содержит значение максимального предела нижней оси графика, и при его изменении во время работы программы автоматически изменяется изображение графика.

9.2. Пример написания программы

З а д а н и е : составить программу, отображающую графики функций sin(x) и cos(x) на интервале [Xmin, Xmax]. Предусмотреть возможность изменения разметки координатных осей, а также шага построения таблицы.

Прежде всего, следует поместить на форму сам элемент управления `Chart`. Он располагается в панели элементов в разделе *Данные*.

Список графиков хранится в свойстве `Series`, который можно изменить, выбрав соответствующий пункт в окне свойств. Поскольку на одном поле требуется вывести два отдельных графика функций, нужно добавить еще один элемент. Оба элемента, и существующий и добавленный, нужно соответствующим образом настроить: изменить тип диаграммы `ChartType` на `Spline`. Здесь же можно изменить подписи к графикам с абстрактных *Series1* и *Series2* на sin(x) и cos(x) – за это отвечает свойство `Legend`. Наконец, с помощью свойства `BorderWidth` можно сделать линию графика потолще, а затем поменять цвет линии с помощью свойства `Color`.

Ниже приведен текст обработчика нажатия кнопки «Расчет!», который выполняет все требуемые настройки и расчеты и отображает графики функций:

```
private void buttonCalc_Click(object sender,
   EventArgs e)
{
   // Считываем с формы требуемые значения
   double Xmin = double.Parse(textBoxXmin.Text);
   double Xmax = double.Parse(textBoxXmax.Text);
   double Step = double.Parse(textBoxStep.Text);

   // Количество точек графика
   int count = (int)Math.Ceiling((Xmax - Xmin) / Step)
        + 1;

   // Массив значений X – общий для обоих графиков
   double[] x = new double[count];

   // Два массива Y – по одному для каждого графика
   double[] y1 = new double[count];
   double[] 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]);
        }

        // Настраиваем оси графика
        chart1.ChartAreas[0].AxisX.Minimum = Xmin;
        chart1.ChartAreas[0].AxisX.Maximum = Xmax;

        // Определяем шаг сетки
        chart1.ChartAreas[0].AxisX.MajorGrid.Interval = Step;

        // Добавляем вычисленные значения в графики
        chart1.Series[0].Points.DataBindXY(x, y1);
        chart1.Series[1].Points.DataBindXY(x, y2);
    }
```

9.3. Выполнение индивидуального задания

Постройте график функции для своего варианта из лабораторной работы № 4. Таблицу данных получить путем изменения параметра X с шагом dx. Добавьте второй график для произвольной функции.

ЛАБОРАТОРНАЯ РАБОТА № 10. КОМПЬЮТЕРНАЯ ГРАФИКА

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

10.1. Событие `Paint`

Для форм в C# предусмотрен 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. 10.1), наполнив его необходимой функциональностью.

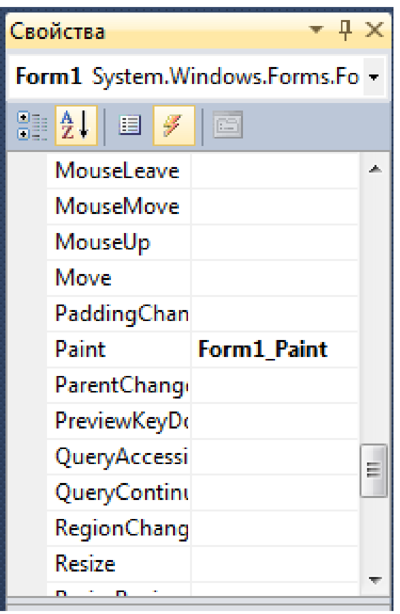


Рис. 10.1. Создание обработчика события Paint

10.2. Объект `Graphics` для рисования

Для рисования линий и фигур, отображения текста, вывода изображений и т. д. нужно использовать объект `Graphics`. Этот объект предоставляет поверхность рисования и используется для создания графических изображений. Ниже представлены два этапа работы с графикой.

- Создание или получение объекта `Graphics`.
- Использование объекта `Graphics` для рисования.

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;
   // Далее вставляется код рисования
}
```

10.3. 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 незакрашенных фигур (таких, например, как прямоугольные рамки), а вторые – для рисования закрашенных геометрических фигур. Ниже рассматривается применение наиболее часто используемых методов, более полную информацию можно найти в документации по Visual Studio.

Метод DrawLine рисует линию, соединяющую две точки с заданными координатами. У метода есть несколько перегруженных версий:

```
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 создается черное перо толщиной 2 пиксела. При создании пера можно 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 варианты метода DrawLine позволяют 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++)
        g.DrawLine(new Pen(Brushes.Black, 2),
            10, 4 * i + 20, 200, 4 * i + 20);
}
```

Здесь мы вызываем метод DrawLine в цикле, рисуя 50 горизонтальных линий (рис. 10.2).

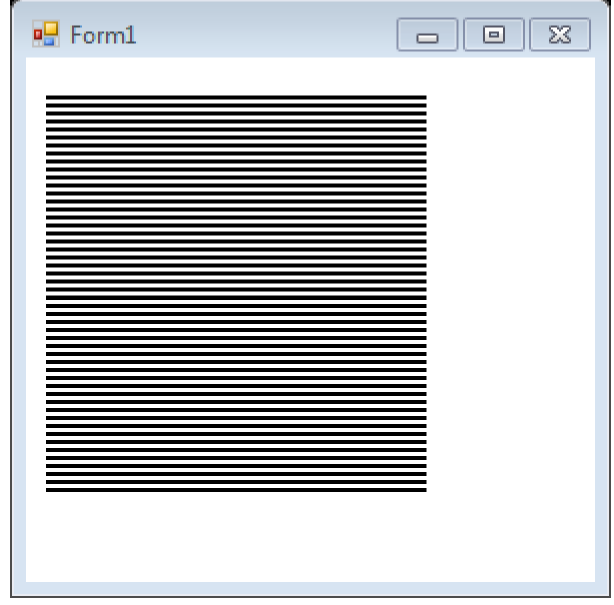


Рис. 10.2. Пример использования DrawLine

Вызвав один раз метод DrawLines, можно нарисовать сразу несколько прямых линий, соединенных между собой. Иными словами, метод DrawLines позволяет соединить между собой несколько точек. Ко-

ординаты этих точек по горизонтальной и вертикальной осям 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. Код будет выглядеть следующим образом:

```
Point[] points = new Point[50];
Pen pen = new Pen(Color.Black, 2);

private void Form1_Paint(object sender,
    PaintEventArgs e)
{
    Graphics g = e.Graphics;
    g.DrawLines(pen, points);
}

private void Form1_Load(object sender, EventArgs e)
{
    for (int i = 0; i < 20; i++)
    {
        int xPos;
        if (i % 2 == 0)
        {
            xPos = 10;
        }
        else
        {
            xPos = 400;
        }
        points[i] = new Point(xPos, 10 * i);
    }
}
```

Результат работы программы приведен на рис. 10.3.

Для прорисовки прямоугольников можно использовать метод DrawRectangle:

```
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 следующий метод:

```
DrawPolygon(Pen, Point[]);
```

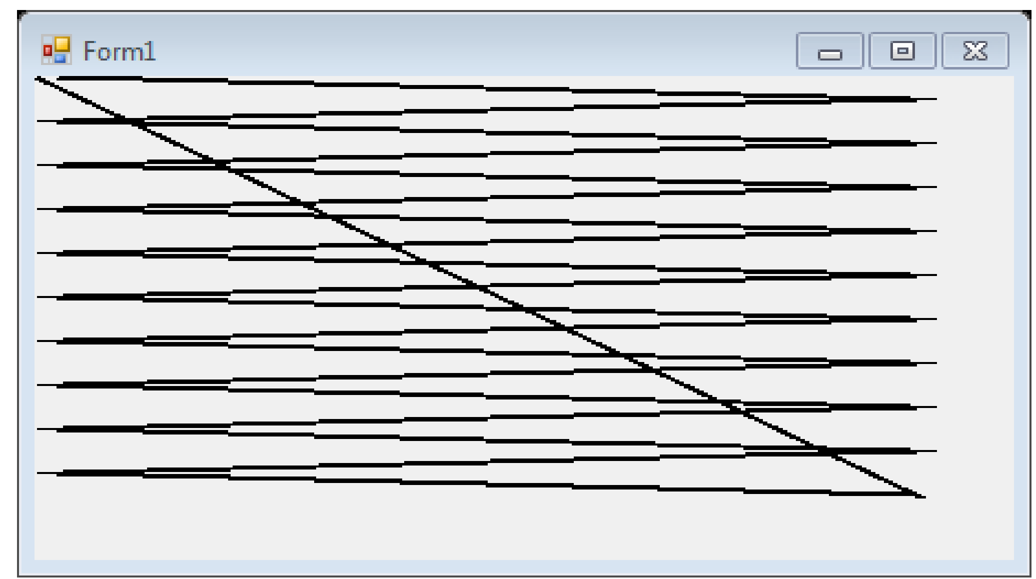


Рис. 10.3. Пример использования массива точек

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 (рис. 10.4).

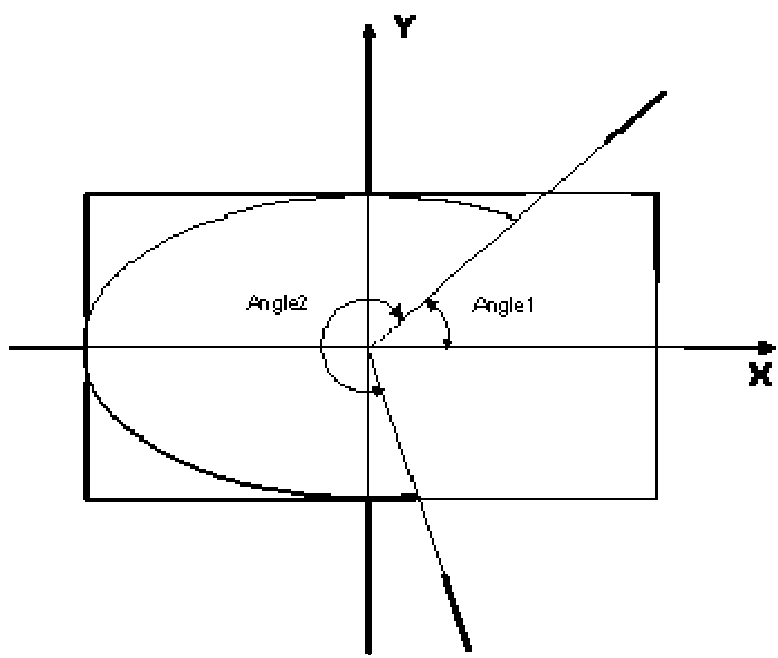


Рис. 10.4. Углы и прямоугольник, задающие сегмент эллипса

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* – незакрашенные. Кроме этого, в качестве первого параметра методам с префиксом Fill* передается не перо класса Pen, а кисть класса SolidBrush. Ниже приведем пример, выводящий закрашенный прямоугольник:

```
SolidBrush B = new SolidBrush(Color.DeepPink);
g.FillRectangle(B, 0, 0, 100, 100);
```

Индивидуальное задание

Изучите с помощью справки MSDN[1] 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).

[1] http://msdn.microsoft.com/ru-ru/library/system.drawing(v=vs.100).aspx

ЛАБОРАТОРНАЯ РАБОТА № 11.
АНИМАЦИЯ

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

11.1. 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 вызываться принудительная перерисовка окна. Для создания анимации весь код, рисующий что-либо на форме, должен находиться в обработчике события `Paint`.

11.2. Создание анимации

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

В качестве примера рассмотрим код анимации секундной стрелки часов:

```
// Глобальные переменные
private int x1, y1, x2, y2, r;
private double a;
private Pen pen = new Pen(Color.DarkRed, 2);

// Перерисовка формы
private void Form1_Paint(object sender,
    PaintEventArgs e)
{
    Graphics g = e.Graphics;
    // Рисуем секундную стрелку
    g.DrawLine(pen, x1, y1, x2, y2);
}
```

```
// Действия при загрузке формы
private void Form1_Load(object sender, EventArgs e)
{
    r = 150; // Радиус стрелки
    a = 0; // Угол поворота стрелки
    // Определяем центр формы – начало стрелки
    x1 = ClientSize.Width / 2;
    y1 = ClientSize.Height / 2;
    // Конец стрелки
    x2 = x1 + (int)(r * Math.Cos(a));
    y2 = y1 - (int)(r * Math.Sin(a));
}
// Действия при очередном «тике» таймера
private void timer1_Tick(object sender, EventArgs e)
{
    a -= 0.1; // Уменьшаем угол на 0,1 радиану
    // Новые координаты конца стрелки
    x2 = x1 + (int)(r * Math.Cos(a));
    y2 = y1 - (int)(r * Math.Sin(a));
    // Принудительный вызов события Paint
    Invalidate();
}
```

11.3. Движение по траектории

Движение по траектории реализуется аналогично вышерассмотренному примеру. Для реализации движения по прямой нужно увеличивать переменные, являющиеся узловыми точками, на определенные константы: в приведенном выше примере это переменные x2 и y2. Для задания более сложной траектории можно использовать различные параметрические кривые.

В случае движения на плоскости обычно изменению подвергается один параметр. Рассмотрим пример реализации движения окружности по *декартову листу*. Декартов лист – это плоская кривая третьего порядка, удовлетворяющая уравнению в прямоугольной системе $x^3 + y^3 = 3 \cdot a \cdot x \cdot y$. Параметр $3 \cdot a$ определяется как диагональ квадрата, сторона которого равна наибольшей хорде петли.

При переходе к параметрическому виду получаем:

$$\begin{cases} x = \frac{3at}{1+t^3} \\ y = \frac{3at^2}{1+t^3}, \end{cases}$$

где $t = \text{tg}\,\varphi$.

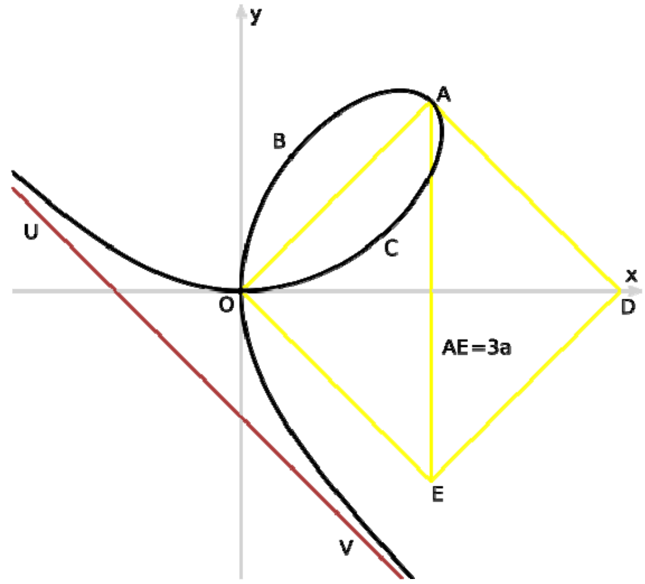


Рис. 11.1. Декартов лист

Описание ряда интересных кривых для создания траектории движения можно найти в Википедии в статье Циклоидальная кривая[2].

Программная реализация выглядит следующим образом:

```
private int x1, y1, x2, y2;
private double a, t, fi;
private Pen pen = new Pen(Color.DarkRed, 2);

private void Form1_Load(object sender, EventArgs e)
{
    x1 = ClientSize.Width / 2;
    y1 = ClientSize.Height / 2;
    a = 150;
    fi = -0.5;
    t = Math.Tan(fi);
    x2 = x1 + (int)((3 * a * t) / (1 + t * t * t));
    y2 = y1 - (int)((3 * a * t * t) / (1 + t * t * t));

}

private void Form1_Paint(object sender,
    PaintEventArgs e)
{
    Graphics g = e.Graphics;
    g.DrawEllipse(pen, x2, y2, 20, 20);
}

private void timer1_Tick(object sender, EventArgs e)
{
```

[2] https://ru.wikipedia.org/wiki/Циклоидальная_кривая

```
        fi += 0.01;
        t = Math.Tan(fi);
        x2 = x1 + (int)((3 * a * t) / (1 + t * t * t));
        y2 = y1 - (int)((3 * a * t * t) / (1 + t * t * t));
        Invalidate();
    }
```

Индивидуальное задание

1. Создайте программу, показывающую пульсирующее сердце.
2. Создайте приложение, отображающее вращающийся винт самолета.
3. Разработайте программу анимации двигающегося человечка.
4. Создайте программу, показывающую движение окружности по синусоиде.
5. Создайте приложение, отображающее движение окружности по спирали.
6. Разработайте программу анимации падения снежинки.
7. Создайте программу, показывающую скачущий мячик.
8. Создайте приложение, отображающее движение окружности вдоль границы окна. Учтите возможность изменения размеров окна.
9. Разработайте программу анимации летающего бумеранга.
10. Создайте программу, показывающую падение нескольких звезд одновременно.
11. Создайте приложение, отображающее хаотичное движение звезды в окне.
12. Разработайте программу анимации взлета ракеты. Старт осуществляется по нажатию специальной «красной» кнопки.
13. Создайте программу, показывающую движение окружности вдоль многоугольника. Число вершин вводится пользователем до анимации.
14. Создайте приложение, отображающее броуновское движение молекулы в окне.
15. Разработайте программу анимации движения планет в Солнечной системе.
16. Создайте программу, показывающую движение квадратика по траектории, состоящей из 100 точек, хранящихся в специальном массиве.
17. Создайте приложение, имитирующие механические часы.
18. Разработайте программу анимации падения нескольких листков с дерева. Движение не должно быть линейным.
19. Создайте программу, показывающую движение окружности по спирали с плавно изменяющейся скоростью.
20. Создайте приложение, отображающее движение автомобиля с вращающимися колесами.

ЛАБОРАТОРНАЯ РАБОТА № 12.
ОБРАБОТКА ИЗОБРАЖЕНИЙ

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

12.1. 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 свойств или в коде программы, указывая на рисунок, который следует отображать.

Элемент управления `PictureBox` содержит и другие полезные свойства, в том числе свойство `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*[3]. 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` – получить доступ к отдельным пикселям рисунка.

---

[3] В приложениях Visual C# часто содержатся данные, неявляющиеся исходным кодом. Такие данные называются ресурсами проекта и могут включать двоичные данные, текстовые файлы, аудио- и видеофайлы, таблицы строк, значки, изображения, XML-файлы или любой другой тип данных, необходимых для приложения. Данные ресурсов проекта хранятся в формате XML в файле с расширением RESX (имя по умолчанию – Resources.resx), который можно открыть в Обозревателе решений.

12.2. Элементы управления `OpenFileDialog` и `SaveFileDialog`

Элемент управления `OpenFileDialog` является стандартным диалоговым окном. Он аналогичен диалоговому окну «Открыть файл» операционной системы Windows. Элемент управления `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 значение `DialogResult.OK` при корректном выборе. Диалоговое окно возвращает путь и имя файла, который был выбран пользователем в специальном свойстве `FileName`.

12.3. 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. 12.1).

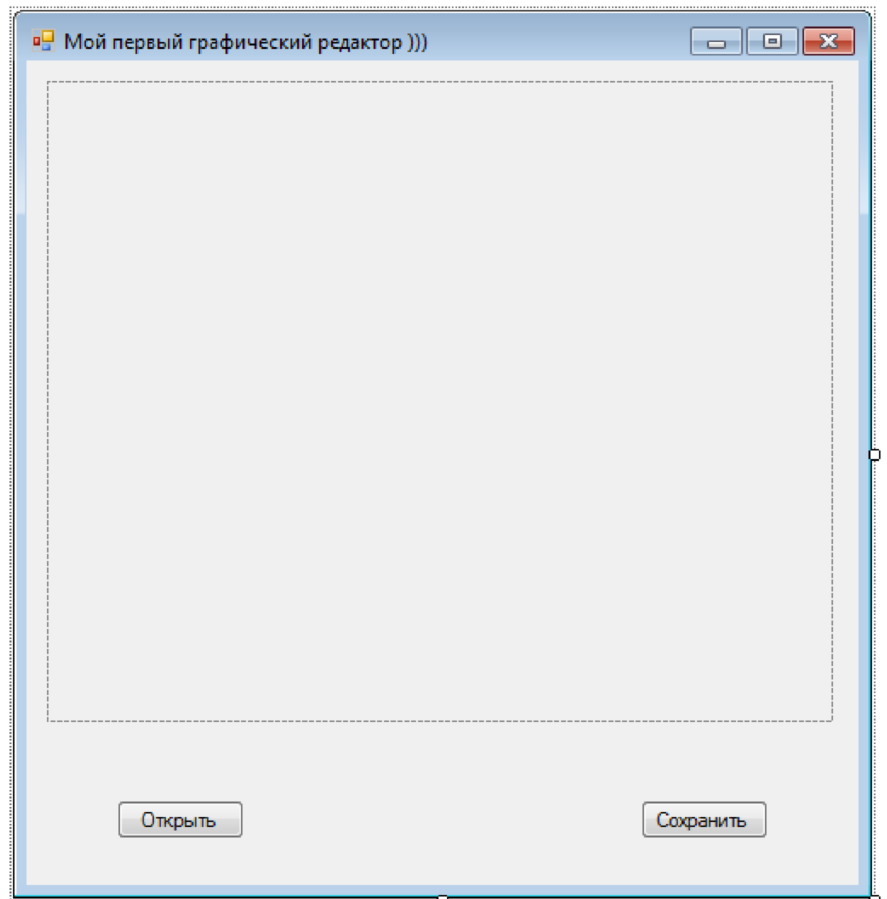


Рис. 12.1. Форма для графического редактора

В этом случае не понадобится из панели элементов размещать на форме элементы 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 вызываться 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.

Код приложения будет выглядеть следующим образом:

```
// Глобальные переменные
private Point PreviousPoint, point;
private Bitmap bmp;
private Pen blackPen;
private Graphics g;

// Действия при загрузке формы
private void Form1_Load(object sender, EventArgs e)
{
   // Подготавливаем перо для рисования
   blackPen = new Pen(Color.Black, 4);
}

// Действия при нажатии кнопки загрузки изображения
private void button1_Click(object sender, EventArgs e)
{
   // Описываем объект класса OpenFileDialog
   OpenFileDialog dialog = new OpenFileDialog();
   // Задаем расширения файлов
   dialog.Filter = "Image files (*.BMP, *.JPG, " +
       "*.GIF, *.PNG)|*.bmp;*.jpg;*.gif;*.png";
   // Вызываем диалог и проверяем выбран ли файл
   if (dialog.ShowDialog() == DialogResult.OK)
   {
   // Загружаем изображение из выбранного файла
       Image image = Image.FromFile(dialog.FileName);
       int width = image.Width;
       int height = image.Height;
       pictureBox1.Width = width;
       pictureBox1.Height = height;
       // Создаем и загружаем изображение в формате bmp
       bmp = new Bitmap(image, width, height);
```

```
        // Записываем изображение в pictureBox1
        pictureBox1.Image = bmp;
        // Подготавливаем объект Graphics для рисования
        g = Graphics.FromImage(pictureBox1.Image);
    }
}

// Действия при нажатии мышки в pictureBox1
private void pictureBox1_MouseDown(object sender,
    MouseEventArgs e)
{
    // Записываем в предыдущую точку текущие координаты
    PreviousPoint.X = e.X;
    PreviousPoint.Y = e.Y;
}

// Действия при перемещении мышки
private void pictureBox1_MouseMove(object sender,
    MouseEventArgs e)
{
    // Проверяем нажата ли левая кнопка мыши
    if (e.Button == MouseButtons.Left)
    {
        // Запоминаем текущее положение курсора мыши
        point.X = e.X;
        point.Y = e.Y;
        // Соеденяем линией предыдущую точку с текущей
        g.DrawLine(blackPen, PreviousPoint, point);
        // Текущее положение курсора - в PreviousPoint
        PreviousPoint.X = point.X;
        PreviousPoint.Y = point.Y;
        // Принудительно вызываем перерисовку
        pictureBox1.Invalidate();
    }
}

// Действия при нажатии кнопки сохранения файла
private void button2_Click(object sender, EventArgs e)
{
    // Описываем и порождаем объект savedialog
    SaveFileDialog savedialog = new SaveFileDialog();
    // Задаем свойства для savedialog
    savedialog.Title = "Сохранить картинку как ...";
    savedialog.OverwritePrompt = true;
    savedialog.CheckPathExists = true;
    savedialog.Filter =
        "Bitmap File(*.bmp)|*.bmp|" +
        "GIF File(*.gif)|*.gif|" +
```

```
            "JPEG File(*.jpg)|*.jpg|" +
            "PNG File(*.png)|*.png";
        // Показываем диалог и проверяем задано ли имя файла
        if (savedialog.ShowDialog() == DialogResult.OK)
        {
            string fileName = savedialog.FileName;
            // Убираем из имени расширение файла
            string strFilExtn = fileName.Remove(0,
                fileName.Length - 3);
            // Сохраняем файл в нужном формате
            switch (strFilExtn)
            {
                case "bmp":
                    bmp.Save(fileName,
                    System.Drawing.Imaging.ImageFormat.Bmp);
                    break;
                case "jpg":
                    bmp.Save(fileName,
                 System.Drawing.Imaging.ImageFormat.Jpeg);
                    break;
                case "gif":
                    bmp.Save(fileName,
                    System.Drawing.Imaging.ImageFormat.Gif);
                    break;
                case "tif":
                    bmp.Save(fileName,
                 System.Drawing.Imaging.ImageFormat.Tiff);
                    break;
                case "png":
                    bmp.Save(fileName,
                    System.Drawing.Imaging.ImageFormat.Png);
                    break;
                default:
                    break;
            }
        }
    }
```

Далее добавим в проект кнопку для перевода изображения в градации серого цвета:

```
    // Действия при нажатии кнопки перевода в градации серого
    private void button3_Click(object sender, EventArgs e)
    {
        // Циклы для перебора всех пикселей на изображении
        for (int i = 0; i < bmp.Width; i++)
            for (int j = 0; j < bmp.Height; j++)
```

```
            {
                // Извлекаем в R значение красного цвета
                int R = bmp.GetPixel(i, j).R;
                // Извлекаем в G значение зеленого цвета
                int G = bmp.GetPixel(i, j).G;
                // Извлекаем в B значение синего цвета
                int B = bmp.GetPixel(i, j).B;
                // Высчитываем среднее арифметическое
                int Gray = (R + G + B) / 3;
                // Переводим число в значение цвета.
                // 255 – показывает степень прозрачности.
                // Остальные значения одинаковы
                Color p = Color.FromArgb(255, Gray, Gray,
                     Gray);
                // Записываем цвет в текущую точку
                bmp.SetPixel(i, j, p);
            }
        // Вызываем функцию перерисовки окна
        Refresh();
    }
```

Данный код демонстрирует возможность обращения к отдельным пикселям. Цвет каждого пикселя хранится в модели RGB и состоит из трех составляющих: красного, зеленого и синего цвета, называемых каналами. Значение каждого канала может варьироваться в диапазоне от 0 до 255.

Индивидуальное задание

Добавьте в приведенный графический редактор свои функции в соответствии с вариантом.

1. Расширьте приложение путем добавления возможности выбора пользователем цвета и величины кисти.
2. Разработайте функцию, добавляющую на изображение 1000 точек с координатами, заданными случайным образом. Цвет также задается случайным образом.
3. Создайте функцию, переводящую изображение в черно-белый формат. Пороговое значение задавать с помощью элемента управления TrackBar.
4. Разработайте функцию, оставляющую на изображении только один из каналов (R, G, B). Канал выбирается пользователем.
5. Создайте функцию, выводящую на изображение окружность. Центр окружности совпадает с центром изображения. Все точки вне окружности переводятся в градации серого цвета. Все точки внутри окружности остаются неизменными. Радиус окружности задается пользователем.

6. Создайте функцию, выводящую на изображение треугольник. Для всех точек вне треугольника оставьте только канал B. Все точки внутри треугольника переводятся в градации серого цвета.

7. Создайте функцию, выводящую на изображение ромб. Все точки вне ромба переводятся в градации серого цвета. Для всех точек внутри ромба оставьте только канал G.

8. Разработайте функцию, которая каждую четную строку изображения переводит в градации серого цвета.

9. Разработайте функцию, которая переводит каждый нечетный столбец пикселей (вертикальные линии) в градации серого цвета.

10. Создайте функцию, разбивающую изображение на четыре равные части. В каждой оставьте значение только одного канала R, G и B, а в четвертой выведите градации серого цвета.

11. Разработайте функцию, заменяющую все точки синего цвета на точки красного цвета.

12. Создайте функцию, инвертирующую изображение в градациях серого цвета в негатив.

13. Создайте функцию, изменяющую яркость изображения, путем прибавления или уменьшения заданной пользователем величины к каждому каналу.

14. Создайте функцию, переводящую изображение в черно-белый формат в соответствии с пороговым значением, которое ввел пользователь. Для анализа используйте только один из каналов (R, G, B).

15. Разработайте функцию для создания эффекта мозаики. При этом изображения разбиваются на прямоугольные фрагменты, в каждом из которых выбирается цвет средней точки, и этим же цветом закрашивается весь фрагмент.

16. Разработайте функцию, разбивающую изображение на фрагменты, в каждом из которых остается только один из каналов (R, G, B).

17. Разработайте функцию, изменяющую значение канала R на всем изображении.

18. Разработайте функцию, обнуляющую значение выбранного пользователем канала на всем изображении.

19. Создайте функцию, переводящую прямоугольную область на изображении в градации серого цвета. Разработайте интерфейс, через который пользователь может задавать координаты этой области.

20. Разработайте и реализуйте собственный алгоритм, переводящий изображение в градациях серого цвета в цвет.

ЛАБОРАТОРНАЯ РАБОТА № 13.
МЕТОДЫ

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

13.1. 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 подчеркивания, 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 имен и т. д. Если метод должен возвращать какой-то ре-

зультат, 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.

Пример метода, вычисляющего выражение:

```
public double Calc(double a, double b, double c)
{
    if (a > b)
        return Math.Sin(a) * Math.Cos(b);
    else
    {
        double k = Math.Tan(a * b);
        return k * Math.Exp(c / k);
    }
}
```

13.2. Перегрузка методов

Язык C# позволяет создавать несколько методов с одинаковыми именами, но разными параметрами. Компилятор автоматически подберет наиболее подходящий метод при построении программы. Например, можно написать два отдельных метода возведения числа в степень: для целых чисел будет применяться один алгоритм, а для вещественных – другой:

```
/// <summary>
/// Вычисление X в степени Y для целых чисел
/// </summary>
private int Pow(int X, int Y)
{
    int b = 1;
    while (Y != 0)
        if (Y % 2 == 0)
        {
            Y /= 2;
            X *= X;
        }
        else
        {
            Y--;
            b *= X;
        }
    return b;
}
```

```
/// <summary>
/// Вычисление X в степени Y для вещественных чисел
/// </summary>
private double Pow(double X, double Y)
{
    if (X != 0)
        return Math.Exp(Y * Math.Log(Math.Abs(X)));
    else if (Y == 0)
        return 1;
    else
        return 0;
}
```

Вызывается такой код одинаково, разница лишь в параметрах – в первом случае компилятор вызовет метод `Pow` с целочисленными параметрами, а во втором – с вещественными:

```
Pow(3, 17);
Pow(3.0, 17.0);
```

13.3. 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. Для этого при реализации метода нужным параметрам следует присвоить значение прямо в списке параметров:

```
private void GetData(int Number, int Optional = 5)
{
    MessageBox.Show("Number: {0}", Number);
    MessageBox.Show("Optional: {0}", Optional);
}
```

В этом случае вызывать метод можно следующим образом:

```
GetData(10, 20);
GetData(10);
```

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 берет 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)
```

13.4. 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);
MessageBox.Show(n.ToString());
```

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

Чтобы метод мог изменять переданную ему переменную, ее 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);
MessageBox.Show(n.ToString());
```

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. передается уже не копия, а ссылка на реальную переменную в памяти.

Если метод использует переменные по ссылке только для возврата значений и не имеет значения, 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);
```

Индивидуальное задание

1. Написать метод min(x, y), находящий минимальное значение из двух чисел. С его помощью найти минимальное значение из четырех чисел a, b, c, d.
2. Написать метод max(x, y), находящий максимальное значение из двух чисел. С его помощью найти максимальное значение из четырех чисел a, b, c, d.
3. Написать метод, вычисляющий значение n/x^n. С его помощью вычислить выражение:

$$\sum_{i=1}^{10} \frac{i}{x^i}.$$

4. Написать метод, вычисляющий значение n/x^n. С его помощью вычислить выражение:

$$\prod_{i=1}^{10} \frac{i}{x^i}.$$

5. Написать метод, вычисляющий значение $x^n/(n+x)$. С его помощью вычислить выражение:

$$\sum_{i=1}^{10} \frac{x^i}{x+i}.$$

6. Написать метод, вычисляющий значение sin(x) + cos(2 * x). С его помощью определить, в какой из точек a, b или c значение будет минимальным.
7. Написать метод, вычисляющий значение $x^2 + y^2$. С его помощью определить, с какой парой чисел (a, b) или (c, d) значение будет максимальным.
8. Написать метод, вычисляющий значение $x^2 * y^3 * \sqrt{z}$. С его помощью определить, с какой тройкой чисел (a, b, c) или (d, e, f) значение будет максимальным.

9. Написать метод, который у четных чисел меняет знак, а нечетные числа оставляет без изменения. С его помощью обработать ряд чисел от 1 до 10.

10. Написать метод, который положительные числа возводит в квадрат, а отрицательные – в куб. С его помощью обработать ряд чисел от –10 до 10.

11. Написать метод, который вычисляет значения $x = \sin^2(a)$ и $y = \cos^2(a)$. Напечатать таблицу значений от $-\pi$ до π с шагом $\pi/4$.

12. Написать метод, который вычисляет значения $x = a^2$ и $y = \sqrt{a}$. Напечатать таблицу значений от –10 до 10 с шагом 1.

13. Написать метод, который в переданной строке заменяет все точки на многоточие. С его помощью обработать пять разных строк и отобразить их на экране.

14. Написать метод, который в переданной строке заменяет все строчные буквы на заглавные, и наоборот. С его помощью обработать пять разных строк и отобразить их на экране.

15. Написать метод, который разделяет переданную строку на две отдельных строки: первая содержит исходную строку до первой точки, а вторая – исходную строку после первой точки. С его помощью обработать пять разных строк и отобразить результаты на экране.

16. Написать метод, который подсчитывает количество знаков препинания в переданной строке. С его помощью обработать пять разных строк и отобразить результаты на экране.

17. Написать метод, который находит сумму чисел в переданной строке. Числом считается непрерывная последовательность цифр, отделенная от остального текста пробелами или расположенная в начале либо конце строки. Допустимо использовать метод `Split` класса `String`. С помощью этого метода обработать пять разных строк и отобразить результаты на экране.

18. Написать метод, определяющий, является ли переданная строка палиндромом, то есть текстом, который слева направо и справа налево читается одинаково без учета пробелов и регистра символов. С помощью этого метода обработать пять разных строк и отобразить результаты на экране.

19. Написать метод, находящий сумму матриц одинакового размера и возвращающий новую матрицу. С помощью этого метода обработать пары матриц и отобразить результаты на экране.

20. Написать метод, находящий сумму элементов, находящихся не на главной диагонали переданной матрицы. С помощью этого метода обработать пары матриц и отобразить результаты на экране.

ЛАБОРАТОРНАЯ РАБОТА № 14.
РЕКУРСИЯ

Цель лабораторной работы: изучить рекурсивные методы, написать программу с использованием рекурсии.

14.1. Общие понятия

Рекурсивным называют метод, если он вызывает сам себя в качестве вспомогательного. В основе рекурсивного метода лежит так называемое *рекурсивное определение* какого-либо понятия. Классическим примером рекурсивного метода является метод, вычисляющий факториал.

Из курса математики известно, что $0! = 1! = 1$, $n! = 1*2*3\ldots*n$. С другой стороны $n! = (n-1)!*n$. Таким образом, известны два частных случая параметра n, а именно $n = 0$ и $n = 1$, при которых мы без каких-либо дополнительных вычислений можем определить значение факториала. Во всех остальных случаях, то есть для $n > 1$, значение факториала может быть вычислено через значение факториала для параметра $n - 1$. Таким образом, рекурсивный метод будет иметь вид:

```
long F(int n)
{
   // Дошли до 0 или 1?
   if (n == 0 || n == 1)
       // Нерекурсивная ветвь
       return 1;
   else
       // Шаг рекурсии: повторный вызов
       // метода с другим параметром
       return n * F(n - 1);
}

// Пример вызова рекурсивного метода
long f = F(3);
MessageBox.Show(f.ToString());
```

Рассмотрим работу описанного выше рекурсивного метода для n = 3.

Первый вызов метода осуществляется из основной программы, в нашем случае командой `f = F(3)`. Этап вхождения в рекурсию обозначим стрелками с подписью «шаг». Он продолжается до тех пор, пока значение переменной `n` не становится равным 1. После этого начинается

выход из рекурсии (стрелки с подписью «возврат»). В результате вычислений получается, что `F(3) = 3 * 2 * 1`.

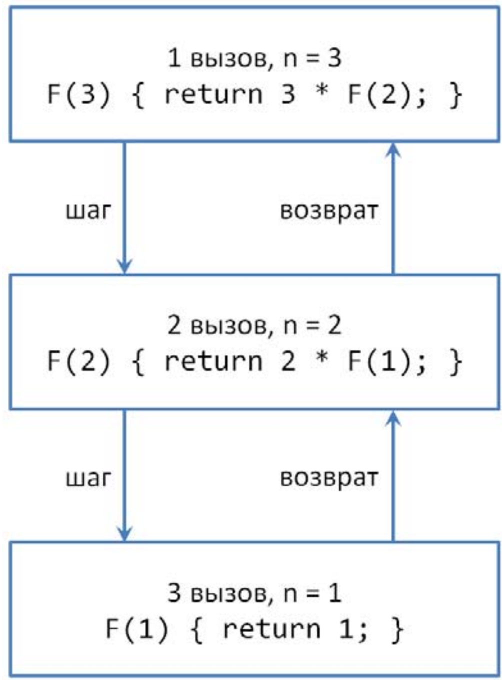


Рис. 14.1. Структура рекурсивных вызовов

Рассмотренный вид рекурсии называют прямой. Метод с прямой рекурсией обычно содержит следующую структуру:

```
if (<условие>)
    <оператор>;
else
    <вызов этого же метода с другими параметрами>;
```

В качестве `<условия>` обычно записываются некоторые граничные случаи параметров, передаваемых рекурсивному методу, при которых результат его работы заранее известен, поэтому далее следует простой оператор или блок, а в ветви `else` происходит рекурсивный вызов данного метода с другими параметрами.

Что необходимо знать для реализации рекурсивного процесса? Со входом в рекурсию осуществляется вызов метода, а для выхода необходимо помнить точку возврата, т. е. то место программы, откуда мы пришли и куда нам нужно будет возвратиться после завершения метода. Место хранения точек возврата называется *стеком вызовов*, и для него выделяется определенная область оперативной памяти. В этом стеке запоминаются не только адреса точек возврата, но и копии значений всех параметров. По этим копиям восстанавливается при возврате вызывающий метод. При развертывании рекурсии за счет создания копий пара-

метров возможно переполнение стека. Это является основным недостатком рекурсивного метода. С другой стороны, рекурсивные методы позволяют перейти к более компактной записи алгоритма.

Следует понимать, что любой рекурсивный метод можно преобразовать в обычный метод с использованием циклов. И практически любой метод можно преобразовать в рекурсивный, если выявить рекуррентное соотношение между вычисляемыми в методе значениями.

Рассмотрим пример кода для создания набора *самоподобных структур*. В нашем случае это будет набор увеличивающихся квадратов (рис. 14.2).

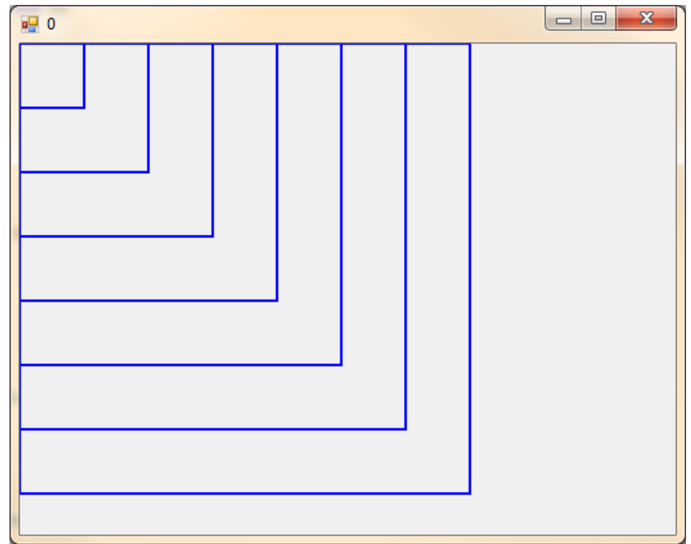


Рис. 14.2. Набор квадратов

При проектировании данной программы были созданы два метода:

```
private void MyDraw(Graphics g, int N, int x, int y)
{
    if (N == 0)
        return;
    else
    {
        // Отрисовка прямоугольника
        g.DrawRectangle(new Pen(Brushes.Blue, 2),
            0, 0, x, y);
        // Увеличение x и y на 50
        x += 50;
        y += 50;
        N--;
        // Рекурсивный вызов с новыми параметрами
        MyDraw(g, N, x, y);
    }
}
```

```
private void Form1_Paint(object sender,
    PaintEventArgs e)
{
    Graphics g = e.Graphics;
    // Первый вызов метода и вход в рекурсию
    MyDraw(g, 7, 50, 50);
}
```

Координаты левого верхнего угла всех прямоугольников неизменны и находятся в точке (0, 0). Поэтому в параметрах метода `MyDraw` достаточно передавать x и y для правого нижнего угла. Также в параметрах передается N, значение которой определяет текущую вложенность рекурсии (сколько вызовов рекурсии еще будет).

14.2. Формирование задержки с помощью таймера

Графические конструкции иногда требуется рассматривать динамически в процессе их построения. Поэтому зачастую используется такая схема вывода графики:

1. Вывод графического элемента.
2. Задержка на n миллисекунд.
3. Повторение 1 и 2 этапа до вывода всех графических элементов.

Реализация задержки с помощью таймера возможна следующим способом:

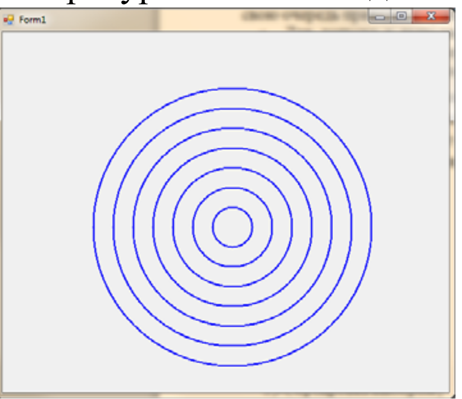
```
// Глобальное поле flag
private bool flag = false;

...
// Далее следует часть программы,
// где необходимо организовать задержку

// Включаем таймер
timer1.Enabled = true;
// Устанавливаем flag в значение true
flag = true;
// Организуем бесконечный цикл
while (flag);
// Выключаем таймер после выхода из цикла
timer1.Enabled = false;

// Обработчик тика таймера
private void timer1_Tick_1(object sender,
    EventArgs e)
{
    // Сбрасываем flag в значение false
    flag = false;
}
```

Идея данного подхода заключается в организации бесконечного цикла, который будет проверять значение некого флага. Однако значение флага может измениться при наступлении события `Tick` таймера, то есть через заданный в таймере промежуток времени. Однако бесконечный цикл, описанный выше, останется бесконечным, и программа просто зависнет. В чем же дело? Дело в том, что при такой организации цикла программа не может опросить очередь сообщений, в которое и будет поступать, в том числе, и событие `Tick` от таймера. Тем самым мы не попадем никогда в обработчик события `timer1_Tick_1`. Чтобы решить данную проблему, надо в теле цикла написать `Application.DoEvents()`, что фактически будет заставлять приложение опрашивать очередь сообщений и в свою очередь приведет к срабатыванию обработчика события `timer1_Tick_1`.

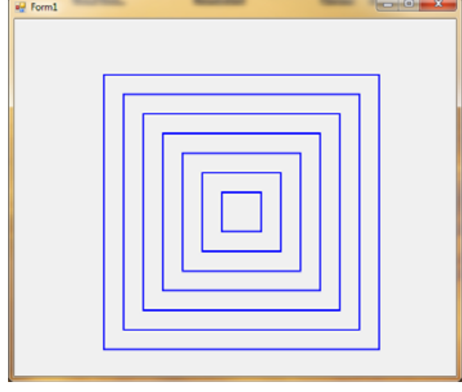
Перед выполнением индивидуального задания по лабораторной работе разработайте приложение, строящее ряд увеличивающихся квадратов (рис. 14.2). Квадраты выводятся последовательно через одну секунду.

Индивидуальное задание

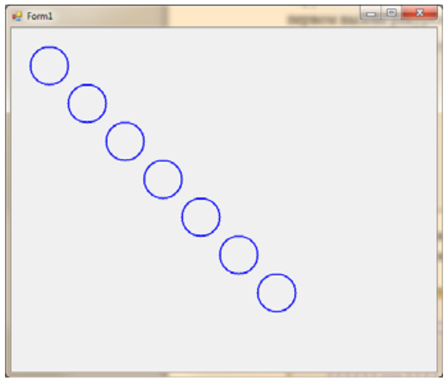
1. Напишите приложение, которое строит ряд окружностей. Центр окружностей совпадает с центром экрана. Число окружностей задается при первом вызове рекурсивного метода.



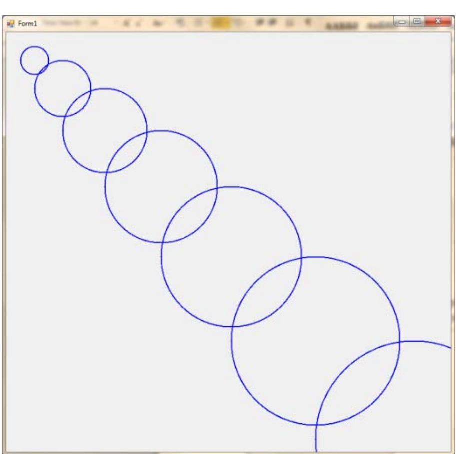
2. Напишите приложение, которое строит ряд квадратов. Центр квадратов совпадает с центром экрана. Число квадратов задается при первом вызове рекурсивного метода.



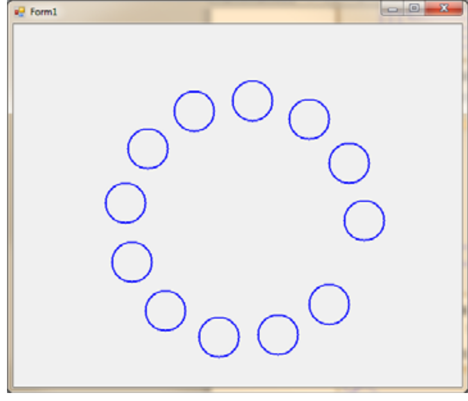
3. Напишите приложение, которое строит ряд окружностей по диагонали. Число окружностей задается при первом вызове рекурсивного метода.



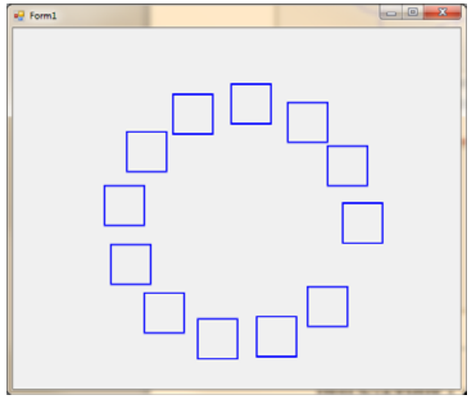
4. Напишите приложение, которое строит ряд увеличивающихся окружностей по диагонали. Число окружностей задается при первом вызове рекурсивного метода.



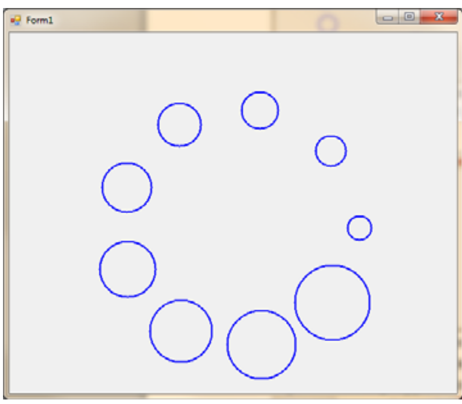
5. Напишите приложение, которое строит ряд окружностей, центры которых лежат на окружности. Число окружностей задается при первом вызове рекурсивного метода.



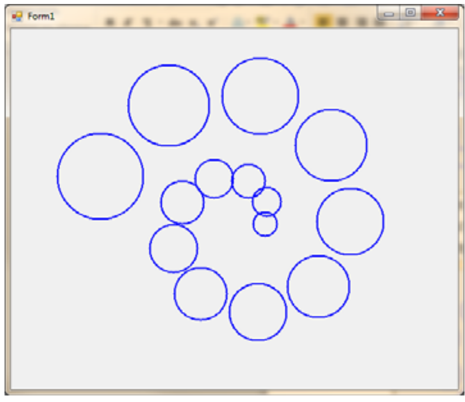
6. Напишите приложение, которое строит ряд квадратов, центры которых лежат на окружности. Число квадратов задается при первом вызове рекурсивного метода.



7. Напишите приложение, которое строит ряд увеличивающихся окружностей, центры которых лежат на окружности. Число окружностей задается при первом вызове рекурсивного метода.



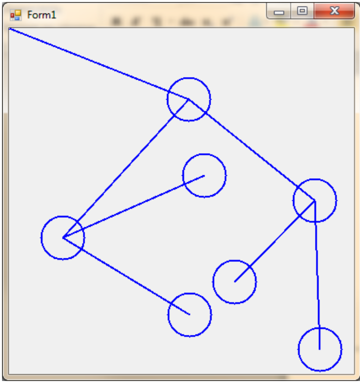
8. Напишите приложение, которое строит ряд увеличивающихся окружностей, центры которых лежат на спирали. Число окружностей задается при первом вызове рекурсивного метода.



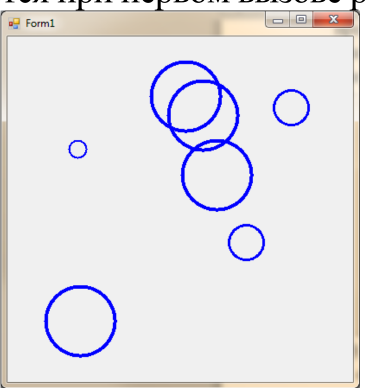
9. Вычислить, используя рекурсию, выражение:

$$\sqrt{6+2\cdot\sqrt{7+3\cdot\sqrt{8+4\cdot\sqrt{9+...}}}}$$

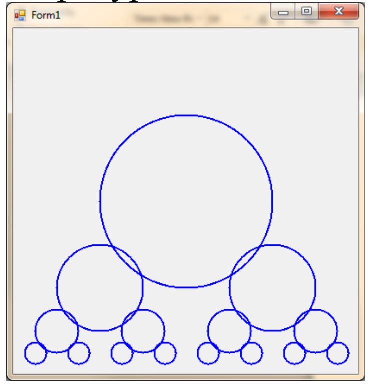
10. Напишите приложение, которое строит ряд окружностей. Число окружностей удваивается на каждом шаге (в рекурсивном методе происходит два рекурсивных вызова). Центры окружностей выбираются каждый раз произвольно (случайно). Линии связывают центры окружностей «предка» и «порожденных» от нее. Число рекурсий задается при первом вызове рекурсивного метода.



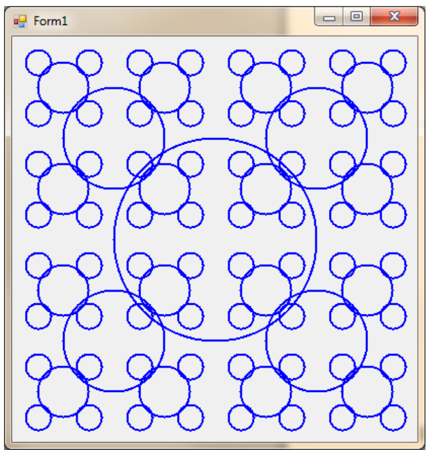
11. Напишите приложение, которое строит ряд увеличивающихся окружностей. Число окружностей удваивается на каждом шаге (в рекурсивном методе происходит два рекурсивных вызова). Центры окружностей выбираются каждый раз произвольно (случайно). Толщина линий также увеличивается. Число рекурсий задается при первом вызове рекурсивного метода.



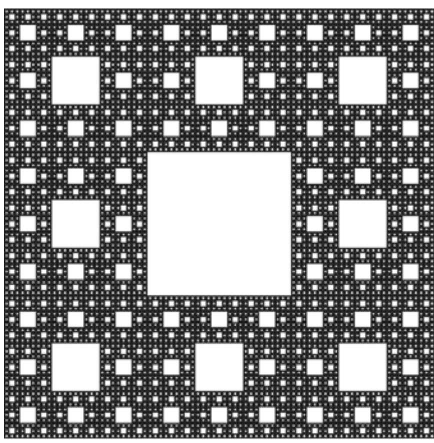
12. Напишите приложение, которое строит ряд уменьшающихся окружностей. Число окружностей удваивается на каждом шаге (в рекурсивном методе происходит два рекурсивных вызова). Число рекурсий задается при первом вызове рекурсивного метода.



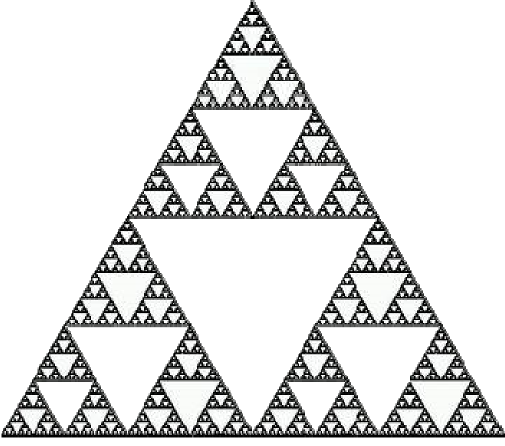
13. Напишите приложение, которое строит приведенное ниже изображение. Число рекурсий задается при первом вызове рекурсивного метода. На каждом шаге число окружностей увеличивается в четыре раза (в рекурсивном методе происходит четыре рекурсивных вызова).



14. Постройте ковер Серпинского.



15. Разработайте программу построения треугольника Серпинского.



16. Реализуйте программу визуализации построения первых n шагов множества Кантора.

17. Реализуйте рекурсивный алгоритм вычисления n-го числа Фибоначчи.

18. Реализуйте рекурсивный алгоритм вычисления n-го факториала.

19. Реализуйте рекурсивный подсчет суммы всех элементов массива. Сумма элементов массива считается по следующему алгоритму: массив делится пополам, подсчитываются и складываются суммы элементов в каждой половине. Сумма элементов в половине массива подсчитывается по тому же алгоритму, то есть снова путем деления пополам. Деления происходят, пока в получившихся кусках массива не окажется по одному элементу и вычисление суммы, соответственно, не станет тривиальным.

20. Дана монотонная последовательность, в которой каждое натуральное число k встречается ровно k раз: 1, 2, 2, 3, 3, 3, 4, 4, 4, 4, … По данному натуральному n выведите первые n членов этой последовательности.

ЛАБОРАТОРНАЯ РАБОТА № 15. СОРТИРОВКА И ПОИСК

Цель лабораторной работы: освоить основные алгоритмы сортировки, написать программу с использованием этих алгоритмов.

15.1. 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 неупорядочен 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.

15.2. 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 элементы массива (без 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;
            }
}
```

15.3. 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 (безпоследнего)
    // 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])
                // Да - теперь это минимальный элемент
                min = j;
        // Минимальный элемент не первый?
        // Меняем местами!
        if (min != i)
        {
```

```
                int t = Array[i];
                Array[i] = Array[min];
                Array[min] = t;
            }
        }
    }
```

15.4. 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:

1. Выбирается некоторый элемент, который называется *опорным*.
2. Реорганизуем массив таким образом, чтобы все элементы, меньшие или равные опорному элементу, оказались слева от него, а все элементы, больше опорного, – справа от него.
3. 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--;
        }
```

```
    } while (i <= j);
    // Рекурсивно выполняем быструю сортировку
    // для массивов слева и справа
    if (Left < j)
        QuickSort(ref Array, Left, j);
    if (i < Right)
        QuickSort(ref Array, i, Right);
}
```

15.5. 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 прекращен, 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++)
        // Нашли нужное значение? Возвращаем его индекс
        if (Array[i] == Value)
            return i;
    // Перебор закончился безрезультатно – возвращаем -1
    return -1;
}
```

Если алгоритм поиска не нашел 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 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
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.

Индивидуальное задание

Общая часть задания: сформировать массив из 100 случайных чисел. Выполнить простой поиск элемента, подсчитать количество итераций. Отсортировать массив всеми рассмотренными методами и посчитать количество итерация для каждого метода. Выполнить поиск элемента методом дихотомии, подсчитать количество итераций. Сделать выводы.

ИНДИВИДУАЛЬНЫЕ ЗАДАНИЯ ПОВЫШЕННОЙ СЛОЖНОСТИ

Для решения геометрических задач повышенной сложности необходимо:

1) знать, как представляются на плоскости такие геометрические объекты, как точка, прямая, отрезок и окружность;
2) уметь находить уравнение прямой, соединяющей две заданные точки;
3) уметь определять координаты точки пересечения двух прямых;
4) знать, как провести перпендикуляр к прямой или определить, являются ли прямые параллельными;
5) уметь находить скалярное и векторное произведения;
6) находить площадь многоугольника;
7) уметь работать с фигурами на плоскости.

Напомним основные моменты, связанные с этими понятиями.

Каждую точку плоскости можно считать вектором с началом в точке (0, 0). Обозначим через $a=(x, y)$ вектор с координатами (x, y). Длина вектора (его модуль) вычисляется по формуле $|a|=\sqrt{x^2+y^2}$.

Скалярное произведение двух векторов – это число, равное произведению модулей этих векторов на косинус угла между ними, $(a,b)=|a|\cdot|b|\cdot\cos\varphi$. Если вектор a имеет координаты (x_1, y_1), а вектор b координаты – (x_2, y_2), то скалярное произведение вычисляется по формуле $(a,b)=x_1\cdot x_2+y_1\cdot y_2$.

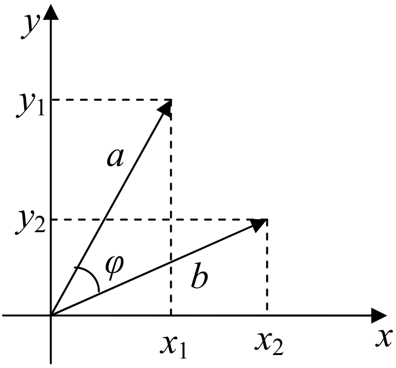


Рис. 16.1. Иллюстрация к скалярному произведению векторов

Заметим, что если угол φ острый, то скалярное произведение $(a, b) > 0$, если угол φ тупой, то $(a, b) < 0$. Если два вектора перпендикулярны, то их скалярное произведение равно нулю.

Векторным произведением двух векторов *a* и *b* называется вектор $[a \times b]$, такой, что

- длина его равна $|[a \times b]| = |a| \cdot |b| \cdot \sin \varphi$;
- вектор $[a \times b]$ перпендикулярен векторам *a* и *b*;
- вектор $[a \times b]$ направлен так, что из его конца кратчайший поворот от *a* к *b* виден происходящим против часовой стрелки.

Длина векторного произведения равна площади параллелограмма, построенного на векторах *a* и *b*.

Через координаты векторов *a* и *b* векторное произведение выражается следующим образом:

$$[a \times b] = \begin{vmatrix} i & j & k \\ x_1 & y_1 & z_1 \\ x_2 & y_2 & z_2 \end{vmatrix} = (y_1 \cdot z_2 - z_1 \cdot y_2)\, i + (x_1 \cdot z_2 - z_1 \cdot x_2)\, j + (x_1 \cdot y_2 - y_1 \cdot x_2)\, k,$$

где *i*, *j*, *k* – единичные вектора осей *Ox*, *Oy*, *Oz* соответственно.

При решении задач на плоскости координаты z_1 и z_2 равны нулю. В этом случае $[a \times b] = (x_1 \cdot y_2 - x_2 \cdot y_1) \cdot k$.

Если вектор *a* образует с осью *Ox* угол α, а вектор *b* – угол β, то для скалярного произведения справедлива формула $[a \times b] = (|a| \cdot |b| \cdot \sin(\beta - \alpha)) \cdot k$. Это означает, что для ненулевых векторов векторное произведение равно нулю тогда и только тогда, когда векторы параллельны. Если поворот от вектора *a* к вектору *b* по наименьшему углу выполняется против часовой стрелки, то $[a \times b] > 0$, если по часовой стрелке, то $[a \times b] < 0$.

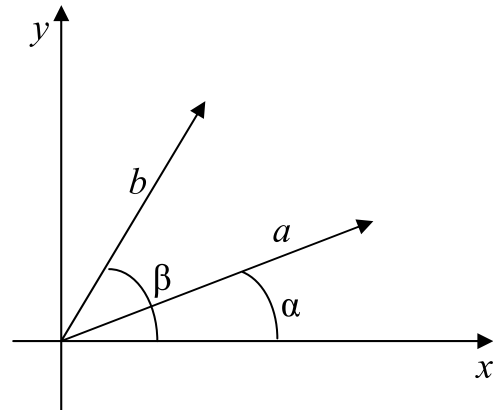


Рис. 16.2. Иллюстрация к векторному произведению

Рассмотрим задачи, при решении которых используются скалярное и векторное произведения.

Задача 1. «Штраф за левые повороты» [1]. *В городе X водителям запрещено выполнять левые повороты. За каждый такой поворот водитель должен уплатить штраф в размере M рублей. Для слежки за водителями в городе установлена компьютерная система, фиксирующая координаты автомобиля в начале движения, в конце движения и во время поворота.*

Исходные данные: N – количество зафиксированных координат автомобиля, (x_i, y_i) – координаты автомобиля в процессе движения, $i = 1, 2, \dots, N$, где (x_1, y_1) – точка начала движения, (x_N, y_N) – последняя точка маршрута автомобиля.

Требуется по заданной последовательности координат движения вычислить сумму штрафа водителя.

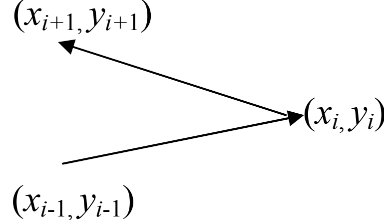


Рис. 16.3. Иллюстрация к задаче «Штраф за левые повороты»

Траекторию движения автомобиля можно представить в виде ломаной, состоящей из направленных отрезков из точек (x_i, y_i) в точки (x_{i+1}, y_{i+1}), $i = 1, 2, \dots, N-1$. Поворот считается левым, если направление текущего отрезка пути a_{i+1} меняется относительно предыдущего отрезка a_i в левую сторону, т. е. против часовой стрелки.

Таким образом, решение задачи сводится к вычислению количества пар участков пути a_i и a_{i+1}, для которых выполняется условие $[a_i \times a_{i+1}] > 0$. Координаты векторов a_i и a_{i+1} вычисляются через координаты точек (x_i, y_i): $a_i = (x_i - x_{i-1}, y_i - y_{i-1})$, $a_{i+1} = (x_{i+1} - x_i, y_{i+1} - y_i)$, следовательно,

$$[a_i \times a_{i+1}] = (x_i - x_{i-1})\,(y_{i+1} - y_i) - (y_i - y_{i-1})(x_{i+1} - x_i),\ i = 2, \dots, N-1.$$

Задача 2. «Здесь будет город-сад». *Жители одного дома города X решили высадить у себя во дворе несколько деревьев. Так как жильцы не смогли договориться, как должны быть расположены посадки, то каждый посадил дерево в том месте двора, где ему захотелось. После проведения посадок полученный сад решили обнести забором. Но пока доски не привезли, деревья обвязали одной длинной веревкой.*

Исходная информация: N – количество деревьев в саду, (x_i, y_i) – координаты деревьев, $i = 1, 2, \dots, N$. Так как были высажены молодые саженцы, то их толщиной можно пренебречь.

Требуется определить, к каким из посаженных деревьев надо привязать веревку так, чтобы все деревья оказались внутри обнесенной зоны, а длина веревки была минимальная.

Эта и подобные ей задачи сводятся к определению для заданного множества точек на плоскости выпуклой оболочки, то есть выпуклого многоугольника с вершинами в некоторых точках из заданного множества, охватывающего все его точки. В [2] приведено несколько вариантов решения такой задачи с учетом временных затрат на выполнение алгоритмов. Здесь мы рассмотрим способ, использующий свойства скалярного произведения векторов.

Будем строить выпуклую оболочку в порядке обхода участка по часовой стрелке. Найдем самую левую точку $M_0 = (x_0, y_0)$, $x_0 = \min\{x_i\}$. Если таких точек несколько, то возьмем самую нижнюю из них. Эта точка наверняка принадлежит искомой выпуклой оболочке. Зададим первоначальный вектор a_0 с началом в точке (x_0, y_0), параллельный оси Oy.

Следующей точкой оболочки будет такая точка M_1, чтобы вектор a_1 с началом в точке M_0 и концом в точке M_1 образовывал с первоначальным вектором a_0 минимальный угол. Если таких точек несколько, то выбирается точка, расстояние до которой максимально.

Далее процесс продолжаем, то есть ищем точку M_2 с минимальным углом между вектором a_1 и вектором a_2 с началом в точке M_1 и концом в точке M_2, затем точку M_3 и т. д. Процесс прекращаем, когда дойдем до первой выбранной точки или количество точек в оболочке станет равно N.

Для определения угла между векторами используется скалярное произведение. Причем сам угол можно не вычислять, так как минимальному углу соответствует максимальный косинус угла.

Задача 3. «Заяц» [3]. *Недалеко от города X находится зоосад. Здешний житель, заяц, хаотично прыгая, оставил след в виде замкнутой самопересекающейся ломаной, охватывающей территорию его владения. Найти площадь минимального по площади выпуклого многоугольника, описанного вокруг этой территории.*

В данной задаче необходимо не только найти выпуклую оболочку множества точек, но и вычислить площадь выпуклого многоугольника с заданным набором вершин.

Исходные данные: N – количество вершин выпуклого многоугольника, (x_i, y_i) – координаты вершин, $i = 1, 2, \ldots, N$.

Требуется определить площадь выпуклого N-угольника.

Площадь N-угольника может быть вычислена как сумма площадей треугольников, из которых N-угольник составлен. Для нахождения площади треугольника используем векторное произведение. Длина векторно-

го произведения векторов, как известно, равна удвоенной площади треугольника, построенного на этих векторах. Пусть вершины треугольника расположены в точках A = (x_1, y_1), B = (x_2, y_2), C = (x_3, y_3). Совместим начало координат с первой точкой. Векторное произведение равно

$$[\mathrm{AB} \times \mathrm{AC}] = \begin{vmatrix} i & j & k \\ x_2 - x_1 & y_2 - y_1 & 0 \\ x_3 - x_1 & y_3 - y_1 & 0 \end{vmatrix},$$

следовательно, площадь треугольника равна

$$S_{ABC} = 1/2\ ((x_2 - x_1)\ (y_3 - y_2) - (y_2 - y_1)(x_3 - x_2)).$$

Значение величины S_{ABC} может быть как положительным, так и отрицательным числом, так как оно зависит от взаимной ориентации векторов AB и AC, поэтому говорят, что площадь ориентированная.

Для нахождения площади N-угольника последний требуется разбить на треугольники и найти сумму ориентированных площадей этих треугольников. Разбиение N-угольника на треугольники можно провести так: зафиксировать одну из вершин N-угольника, например первую, $A_1 = (x_1, y_1)$ и рассматривать все треугольники $A_1A_i\,A_{i+1}$, $i = 2, 3,\ldots, N-1$.

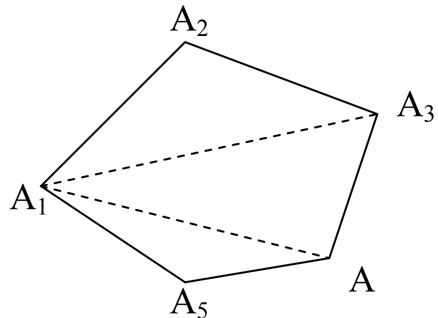


Рис. 16.4. Иллюстрация к задаче «Заяц»

Заметим, что аналогичным способом можно находить площадь произвольного многоугольника.

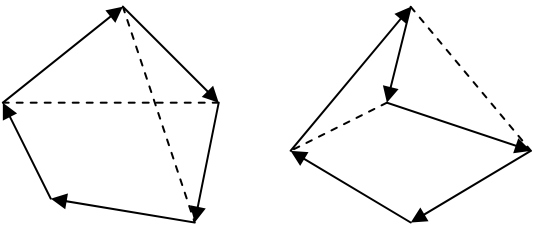


Рис. 16.5. Нахождение площади произвольного многоугольника

Использование свойства ориентации площади треугольника, вычисленной по векторному произведению, позволяет определить, является ли заданный многоугольник выпуклым. Для выпуклого многоугольника все треугольники, образованные тройками соседних вершин в порядке их обхода, имеют одну ориентацию. Поэтому проверка многоугольника на выпуклость может быть проведена с помощью последовательного сравнения знаков векторных произведений для всех пар соседних сторон многоугольника.

Задача 4. «Тигр в загоне». *Недалеко от города X находится заповедник, в котором обитают уссурийские тигры. Работники заповедника очень переживают, когда тигр покидает охраняемую зону. Программа охраны уссурийских тигров предусматривает снабжение каждого тигра ошейником с радиомаяком. Сигнал от тигриного радиомаяка поступает в центр охраны и позволяет определить местоположение тигра. Территория заповедника представляет собой произвольный многоугольник.*

Исходные данные: N – количество вершин многоугольника, задающего заповедник, (x_i, y_i) – координаты его вершин, $i = 1, 2, \ldots, N$. (X, Y) – координаты точки, в которой находится тигр.

Требуется определить, находится ли тигр на территории заповедника, или надо срочно снаряжать спасательную экспедицию.

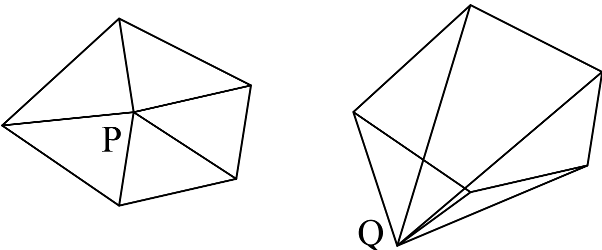


Рис. 16.6. Иллюстрация к задаче «Тигр»

Очень часто при решении задач геометрического содержания требуется проверить, лежит ли заданная точка внутри или вне многоугольника. Таким способом можно решить, например, задачу о Бармаглоте, проверяя каждую точку Бармаглота относительно одеяла-многоугольника. Есть много способов проверки принадлежности точки многоугольнику, однако мы приведем здесь один из них, основанный на использовании произведения векторов.

Идея метода заключается в том, чтобы определить сумму углов, под которыми стороны многоугольника видны из проверяемой точки. Если точка лежит внутри многоугольника, то суммарный угол равен 2π

(точка P на рисунке), если же точка лежит вне многоугольника, то сумма углов не равна 2π (точка Q).

Таким образом, для решения надо перебрать в цикле последовательно все вершины многоугольника и найти сумму углов между векторами PA_i и PA_{i+1}, $i = 1, 2, \ldots, N$. Не забудьте добавить угол между векторами PA_N и PA_1. Для определения величины угла между векторами нам потребуется формула скалярного произведения.

Так как стороны многоугольника должны рассматриваться последовательно, в порядке обхода вершин, то при нахождении суммарного угла следует учитывать взаимное расположение векторов. Угол, под которым сторона видна из исследуемой точки, может быть как положительным, так и отрицательным. Для определения знака угла воспользуемся векторным произведением. Знак векторного произведения и определит знак конкретного угла в общей сумме.

Задача 5. «Пересечение отрезков». Дано n отрезков. Реализовать программу, находящую все их пересечения между собой. Отобразить решение графически.

На плоскости заданы два отрезка a и b, a – точками $A^1(A^1_x, A^1_y)$ и $A^2(A^2_x, A^2_y)$, b – точками $B^1(B^1_x, B^1_y)$ и $B^2(B^2_x, B^2_y)$. Найти и напечатать возможную точку их пересечения $C(C_x, C_y)$. Рассмотрим первый отрезок a. Уравнение прямой, на которой он лежит, можно записать так:

$$x_a = A^1_x + t_a (A^2_x - A^1_x)$$
$$y_a = A^1_y + t_a (A^2_y - A^1_y)$$

Здесь $A^1_x, A^1_y, A^2_x, A^2_y$ – константы, x_a, y_a – точки, принадлежащие отрезку, при t_a изменяющемся от 0 до 1. Аналогично для отрезка b:

$$x_b = B^1_x + t_b (B^2_x - B^1_x)$$
$$y_b = B^1_y + t_b (B^2_y - B^1_y)$$

Таким образом, приравнивая соответствующие координаты, получаем задачу нахождения параметров t_a, t_b, при которых бы выполнялись равенства:

$$A^1_x + t_a (A^2_x - A^1_x) = B^1_x + t_b (B^2_x - B^1_x)$$
$$A^1_y + t_a (A^2_y - A^1_y) = B^1_y + t_b (B^2_y - B^1_y)$$

После разрешения системы относительно t_a, t_b получаем:

$$t_a (A^1_x - A^2_x) + t_b (B^2_x - B^1_x) = A^1_x - B^1_x$$
$$t_a (A^1_y - A^2_y) + t_b (B^2_y - B^1_y) = A^1_y - B^1_y$$

А это есть система из двух линейных уравнений относительно t_a, t_b.

Известно, что система:

$$a_1 x + b_1 y = c_1$$
$$a_2 x + b_2 y = c_2$$

имеет следующее решение:

$x = d_x/d$

$y = d_y/d$,

где d – определитель матрицы,

$d = a_1b_2 - a_2b_1$,

$d_x = c_1b_2 - c_2b_1$,

$d_y = a_1c_2 - a_2c_1$.

В нашей системе относительно t_a, t_b:

$a_1 = A^1_x - A^2_x$

$b_1 = B^2_x - B^1_x$

$c_1 = A^1_x - B^1_x$

$a_2 = A^1_y - A^2_y$

$b_2 = B^2_y - B^1_y$

$c_2 = A^1_y - B^1_y$

откуда легко находится d, d_x, d_y. Если d отличен от нуля, то система имеет единственное решение. Правда, следует помнить, что искомые t_a, t_b параметрически задают отрезки, только если они лежат в диапазоне [0,1], в противном случае – точка пересечения прямых, на которых лежат отрезки, находится вне этих самых отрезков.

Если d равен нулю, а хотя бы один из d_x, d_y отличен от нуля, то отрезки лежат на параллельных прямых, или, как говорят математики, они коллинеарны. Если же все три d, d_x, d_y равны нулю, то это значит, что отрезки лежат на одной и той же прямой, где опять возможны три случая – либо отрезки не перекрываются, либо перекрываются в одной точке, либо перекрываются в бесконечном множестве точек.

Решение ряда задач повышенной сложности опирается на методы, рассмотренные в комбинаторике, а именно на возможность генерации: сочетаний, перестановок, размещений и перечислений элементов.

Одним из важных элементов комбинаторики являются **перестановки**. **Перестановки без повторений** – комбинаторные соединения, которые могут отличаться друг от друга лишь порядком входящих в них элементов. Число таких перестановок определяется как n!.

Для числа 3 количество перестановок будет равно $3! = 3 * 2 * 1 = 6$. Для четырех: $4! = 4 * 3 * 2 * 1 = 24$.

Часто для генерации перестановок используется алгоритм Дейкстры для получения всех перестановок по алфавиту. Разберем этот алгоритм.

Пусть у нас есть первая перестановка (например, 1234). Для нахождения следующей перестановки выполняем три шага.

1. Двигаясь с предпоследнего элемента перестановки, ищем элемент a[i], удовлетворяющий неравенству $a[i] < a[i + 1]$. Для перестановки 1234, это число 3, т. к. $(3 < 4)$.

2. Меняем местами элемент a[i] с наименьшим элементом, который:

а) находится правее a[i];

б) больше, чем a[i].

В нашем случае меняем 3 и 4.

3. Все элементы, стоящие за a[i], сортируем. В нашем случае нужно отсортировать число 4, но это единственный элемент, следовательно, так его и оставляем.

В результате выполнения этих трех шагов получаем следующую по алфавиту перестановку 1243.

Выполнять эти шаги нужно циклически до тех пор, пока в перестановке не будет находиться искомый в первом шаге элемент a[i], т. е. пока перестановка не станет отсортированной по убыванию: 4321.

Перестановки с повторениями – комбинаторные соединения, в которых среди образующих элементов имеются одинаковые. В таких соединениях участвуют несколько типов объектов, причем имеется некоторое количество объектов каждого типа. Поэтому в выборках встречаются одинаковые элементы.

Рис. 16.7. Иллюстрация к задаче «Гномики»

Задание 6. Гномики. Гномики решили встречать гостей с разноцветными шарами. Раздай шарики гномикам так, чтобы цвет шарика не был такой же, как цвет колпачка, и чтобы у гномиков в одинаковых

по цвету колпачках были шарики разного цвета и разной формы. Напишите программу, выводящую все возможные варианты раздачи шариков.

Задание 7. Имеются цифры от 1 до 9, расположенные по возрастанию (убыванию). Требуется расставить между ними произвольное количество знаков <<плюс>> и <<минус>>, чтобы получилось выражение со значением 100. Например,

$123 + 4 - 5 + 67 - 89 = 100$

$9 - 8 + 76 - 5 + 4 + 3 + 21 = 100$

Найти все возможные варианты таких выражений.

Задача 8. Дан двумерный массив, заполненный нулями и единицами. Найти прямоугольник наибольшей площади, заполненный единицами.

Площадь прямоугольников изменяется от максимальной (весь массив) до минимальной (прямоугольник, состоящий из одной 1). Каждый прямоугольник конкретной площади может быть построен множеством способов. Для площади S допустимый прямоугольник – это такой, произведение сторон которого равно S. Мы должны для каждого значения площади перебрать все допустимые способы построения прямоугольников. Каждый прямоугольник конкретной площади и формы может располагаться в массиве различным образом. Точнее сказать, его левая верхняя вершина может находиться в разных точках массива. Следовательно, для прямоугольника определенной площади и формы мы должны перебрать все возможные расположения.

Может показаться, что программа для большого массива будет работать слишком долго, но есть серьезные возможности для ее ускорения. А именно:

1. Если площадь перебирать от максимальной к минимальной, то первый найденный прямоугольник и будет искомым.
2. Прямоугольник конкретной площади и формы не поместится в любом положении в массив.

Учет этих утверждений ведет к очень серьезному ускорению программы.

Задание 9. «Вирус». Колония клеток представляет собой квадратную матрицу порядка N (N < 500). В колонию проникает M (M < 11) вирусов, которые поражают клетки с координатами (X1, Y1), … (Xm, Ym). За одну единицу времени вирус проникает в клетки, соседние с зараженными (соседними считаются клетки, имеющие общую сторону). Требуется написать программу, которая определит время заражения всей колонии. Графически показать процесс заражения.

Задание 10. «Сундук Билли Бонса». Билли Бонс положил в сундук некоторое количество золотых монет. На второй год он вынул из сундука сколько-то монет. Начиная с третьего года, он добавлял столько монет, сколько было в сундуке два года назад.

Требуется написать программу, определяющую количество монет в сундуке в первый и во второй года, если в X-м году там оказалось ровно Y монет. (3 <= X <= 20) и Y (1 <= Y <= 32767).

Пояснение: если в первый год положить 5 монет, а во второй год вынуть 3 монеты, то, начиная с первого года, в сундуке будет 5, 2, 7, 9, 16, 25, … монет.

ПРИЛОЖЕНИЕ 1. СВОЙСТВА ЭЛЕМЕНТОВ УПРАВЛЕНИЯ

Многие стандартные визуальные элементы управления имеют одинаковые свойства. Поэтому имеет смысл рассмотреть их отдельно.

`Name` — Возвращает или задает имя элемента управления. Значение этого свойства используется в программе для обращения к объекту по его имени.

`Size` — Возвращает или задает размер элемента управления. Это свойство позволяет одновременно установить высоту и ширину (в точках) вместо того, чтобы устанавливать по отдельности свойства `Height` и `Width`.

`Height` — Возвращает или задает высоту элемента управления.

`Width` — Возвращает или задает ширину элемента управления.

`Location` — Возвращает или задает координаты левого верхнего угла элемента управления относительно левого верхнего угла контейнера.

`Dock` — Используется для определения способа автоматического изменения размеров элемента управления при изменении размеров родительского элемента управления. Например, задание для свойства `Dock` значения `DockStyle.Left` приводит к выравниванию самого элемента управления по левому краю его родительского элемента управления и к изменению размеров при изменении размеров родительского элемента управления.

Внимание: свойства `Anchor` и `Dock` являются взаимоисключающими. Одновременно может быть задано только одно из них, которое и получает преимущество.

`Anchor` — Возвращает или задает границы контейнера, с которым связан элемент управления, и определяет способ изменения размеров элемента управления при изменении размеров его родительского элемента. Элемент управления можно привязать к одной или нескольким границам кон-

тейнера. Например, если имеется объект `Form` с объектом `Button`, для свойства `Anchor` которого заданы значения `Top` и `Bottom`, то объект `Button` растягивается, чтобы сохранить закрепленное расстояние до верхней и нижней границ объекта `Form` при увеличении значения свойства `Height` объекта `Form`.

Внимание: Свойства `Anchor` и `Dock` являются взаимоисключающими. Одновременно может быть задано только одно из них, которое и получает преимущество.

`Margin`	Возвращает или задает пустое пространство между элементами управления. Элементы управления получают для свойства `Margin` значения по умолчанию, которые достаточно близки к рекомендациям по пользовательскому интерфейсу Windows. Для конкретных приложений по-прежнему могут быть необходимы некоторые корректировки.
`BackColor`	Возвращает или задает цвет фона для элемента управления. Свойство `BackColor` является внешним свойством.
`ForeColor`	Получает или задает основной цвет элемента управления. Свойство `ForeColor` является внешним свойством.
`Font`	Возвращает или задает шрифт текста, отображаемого элементом управления. Нельзя поменять отдельные элементы свойства `Font` – можно только создать новый объект `Font` с требуемыми параметрами и назначить его свойству `Font`. Свойство `Font` является внешним свойством. Внешнее свойство – это свойство элемента управления, которое (если оно не задано) получается из родительского элемента управления.
`Text`	Получает или задает текст, сопоставленный с этим элементом управления. Свойство `Text` элемента управления по-разному используется каждым производным классом. Например, свойство `Text` объекта `Form` отображается в заголовке окна в верхней части формы, содержит небольшое количество символов и, как правило, отображает имя приложения или документа. Однако свойство `Text` объекта `RichTextBox` может быть большим и включать в себя многочисленные невидимые символы, применяемые для фор-

матирования текста. Например, отображаемый в объекте `RichTextBox` текст можно отформатировать, настроив свойства `Font` либо добавив символы пробелов или табуляции для выравнивания текста.

`TextAlign` Получает или задает выравнивание текста для элемента управления.

`Enabled` Возвращает или задает значение, показывающее, сможет ли элемент управления отвечать на действия пользователя. Значение `true`, если элемент управления может отвечать на действия пользователя; в противном случае – значение `false`. Значением по умолчанию является `true`. С помощью свойства `Enabled` можно включать или отключать элементы управления во время выполнения. Например, можно отключить элементы управления, не применяемые при данном состоянии приложения. Можно также отключить элемент управления, чтобы ограничить его использование. Например, возможно отключить кнопку, чтобы пользователь не смог ее нажать. Если элемент управления отключен, его невозможно выделить.

`Visible` Получает или задает значение, указывающее, отображаются ли элемент управления и все его дочерние элементы управления. Значение `true`, если элемент управления и все его дочерние элементы управления отображаются; в противном случае – значение `false`. Значение по умолчанию – `true`. Обратите внимание, что даже если для `Visible` задано значение `true`, элемент управления может быть невидимым для пользователя, если он находится позади других элементов управления.

`Items` С помощью этого свойства можно получить ссылку на список элементов, хранящихся в настоящее время в элементе управления (например, `ListBox`). С помощью этой ссылки можно добавлять и удалять элементы, а также определять число элементов в коллекции.

ПРИЛОЖЕНИЕ 2. СОБЫТИЯ ЭЛЕМЕНТОВ УПРАВЛЕНИЯ

Событие	Описание
`Load`	Происходит до первоначального отображения элемента управления (обычно формы).
`Resize`	Происходит при изменении размеров элемента управления (например, формы).
`Move`	Происходит при перемещении элемента управления.
`Click`	Происходит при щелчке элемента управления. Событие `Click` передает объект `EventArgs` его обработчику событий, указывая только, что щелчок был выполнен. Если необходимы более точные сведения о мыши (кнопка, количество щелчков, вращение колесика или положение), следует использовать событие `MouseClick`. Однако событие `MouseClick` не возникает, если щелчок был выполнен не с помощью мыши, а, например, при нажатии клавиши *Enter*.
`DoubleClick`	Происходит, когда элемент управления дважды щелкается. Двойной щелчок определяется параметрами мыши в операционной системе пользователя. Пользователь может задать время между нажатиями кнопки мыши, которые будут считаться двойным щелчком, а не двумя отдельными щелчками. Событие `Click` вызывается каждый раз, когда элемент управления дважды щелкается. Например, при наличии обработчиков для событий `Click` и `DoubleClick` объекта `Form` события `Click` и `DoubleClick` вызываются, когда форма дважды щелкается и оба метода вызываются. Если элемент управления дважды щелкается и этот элемент управления не поддерживает событие `DoubleClick`, событие `Click` может быть вызвано дважды.
`MouseClick`	Происходит при щелчке элемента управления мышью. Если нажать кнопку мыши, когда курсор находится на элементе управления, обычно возникает

	следующая последовательность событий, относящихся к этому элементу управления: 1. Событие `MouseDown`. 2. Событие `Click`. 3. Событие `MouseClick`. 4. Событие `MouseUp`.
`MouseDoubleClick`	Генерируется при двойном щелчке элемента управления мышью. Событие `MouseDoubleClick` происходит, когда пользователь быстро дважды нажимает кнопку мыши, когда курсор находится на элементе управления. Интервал времени, позволяющий отличить два отдельных щелчка мыши от двойного щелчка, определяется параметрами мыши в операционной системе. При выполнении пользователем такого действия элемент управления вызывает следующую последовательность событий: 1. Событие `MouseDown`. 2. Событие `Click`. 3. Событие `MouseClick`. 4. Событие `MouseUp`. 5. Событие `MouseDown`. 6. Событие `DoubleClick`. 7. Событие `MouseDoubleClick`. 8. Событие `MouseUp`.
`MouseDown`	Происходит при нажатии кнопки мыши, если указатель мыши находится на элементе управления.
`MouseUp`	Происходит при отпускании кнопки мыши, когда указатель мыши находится на элементе управления.
`MouseMove`	Происходит при перемещении указателя мыши по элементу управления. Обычно использование события `MouseMove` приводит к изменению цвета элемента управления или к прорисовке приподнятого прямоугольника вокруг элемента управления.
`MouseLeave`	Происходит, когда указатель мыши покидает элемент управления.

`KeyPress`	Происходит при нажатии клавиши, если элемент управления имеет фокус. Событие `KeyPress` вызывается только нажатием клавиш с символами. Остальные клавиши вызывают события `KeyDown` и `KeyUp`. Свойство `KeyChar` используется для выбора образцов нажатий клавиш во время выполнения и для использования или изменения подмножества стандартных нажатий клавиш. Чтобы обрабатывать события клавиатуры только на уровне формы без предоставления другим элементам управления возможности получать события клавиатуры, необходимо задать для свойства `KeyPressEventArgs.Handled` в методе обработки события `KeyPress` формы значение `true`. События нажатия клавиши происходят в следующем порядке. 1. `KeyDown` 2. `KeyPress` 3. `KeyUp`
`KeyDown`	Происходит при нажатии клавиши, если элемент управления имеет фокус. Чтобы обрабатывать события клавиатуры только на уровне формы без предоставления другим элементам управления возможности получать события клавиатуры, необходимо задать для свойства `KeyPressEventArgs.Handled` в методе обработки события `KeyPress` формы значение `true`. Некоторые клавиши, такие как *Tab*, *Enter*, *Escape* и клавиши со стрелками, автоматически обрабатываются элементами управления.
`KeyUp`	Происходит, когда отпускается клавиша, если элемент управления имеет фокус.
`Enter`	Происходит при входе в элемент управления (при получении фокуса). Когда выполняется изменение фокуса с помощью клавиатуры (*Tab*, *Shift*+*Tab* и т. д.), события фокуса происходят в следующем порядке: 1. `Enter` 2. `GotFocus` 3. `Leave`

	4. Validating 5. Validated 6. LostFocus При изменении фокуса с помощью мыши или посредством вызова метода Focus события фокуса возникают в следующем порядке. 1. Enter 2. GotFocus 3. LostFocus 4. Leave 5. Validating 6. Validated События Enter и Leave подавляются классом Form. В классе Form им эквивалентны события Activated и Deactivate. Не пытайтесь задать фокус из обработчиков событий Enter, GotFocus, Leave, LostFocus, Validating или Validated. Это может привести к тому, что приложение перестанут отвечать.
Leave	Происходит, когда фокус ввода покидает элемент управления. События Enter и Leave подавляются классом Form. В классе Form им эквивалентны события Activated и Deactivate.
TextChanged	Происходит при изменении значения свойства Text. Данное событие возникает в том случае, если свойство Text изменено программой или в результате действий пользователя.
Paint	Происходит при перерисовке элемента управления.

ПРИЛОЖЕНИЕ 3. МЕТОДЫ ДЛЯ РАБОТЫ СО СТРОКАМИ

`Compare()`	Сравнивает две строки и возвращает целое число, которое показывает их относительное положение в порядке сортировки. Возвращаемое число будет равно нулю, если значения параметров равны.
`Concat()`	Соединяет в одну строку две и более строки. При этом разделители не добавляются.
`Copy()` `CopyTo()`	Методы `Copy` и `CopyTo` служат для копирования строки или подстроки в другую строку или в массив типа `Char`.
`Format()`	Форматирует строку, используя строго заданный формат. Для этого заменяет каждый элемент формата в указанной строке текстовым эквивалентом значения соответствующего объекта.
`Join()`	Конкатенация (соединение) массива строк в единую строку. При конкатенации между элементами массива вставляются разделители. Операция, заданная методом `Join`, является обратной к операции, заданной методом `Split`.
`Length`	Свойство, которое возвращает количество символов в строке.
`EndsWith()`	Проверяет, заканчивается ли строка определенной последовательностью символов.
`Insert()`	Вставляет новую строку в уже существующую.
`LastIndexOf()`	Возвращает индекс последнего вхождения элемента в строку.
`PadLeft()`	Выравнивает строку по правому краю, пропуская все пробелы или другие специально заданные символы.
`PadRight()`	Выравнивает строку по левому краю, пропуская все пробелы или другие специально заданные символы.

`Remove()`	Удаляет заданное число символов из строки.
`Replace()`	Заменяет подстроку в заданной позиции на новую подстроку.
`Split()`	Возвращает подстроку, отделенную от основного массива определенным символом. На вход методу `Split` передается один или несколько символов, интерпретируемых как разделители. Объект `string`, вызвавший метод, разделяется на подстроки, ограниченные этими разделителями. Из этих подстрок создается массив, возвращаемый в качестве результата метода. Другая реализация позволяет ограничить число элементов возвращаемого массива.
`StartsWith()`	Определяет, начинается ли строка с определенной последовательности символов.
`Substring()`	Извлекает подстроку из строки.
`ToCharArray()`	Копирует символы из строки в массив символов.
`ToLower()`	Преобразует символы в строке к нижнему регистру.
`ToUpper()`	Преобразует символы в строке к верхнему регистру.
`Trim()`	Удаляет все вхождения определенных символов в начале и конце строки.
`TrimEnd()`	Удаляет все вхождения определенных символов в конце строки.
`TrimStart()`	Удаляет все вхождения определенных символов в начале строки.

ПРИЛОЖЕНИЕ 4. МЕТОДЫ ДЛЯ РАБОТЫ С МАССИВАМИ

`Concat()` Объединяет две последовательности.

`Contains()` Определяет, содержится ли указанный элемент в массиве.

`CopyTo()` Копирует все элементы текущего массива в заданный массив.

`GetLength()` Получает 32-разрядное целое число, представляющее количество элементов в заданном измерении массива. Примером метода `GetLength` может служить метод `GetLength(0)`, который возвращает число элементов в первом измерении массива (например, количество строк в двухмерном массиве).

`Intersect()` Находит пересечение множеств, представленных двумя массивами.

`Length` Свойство, которое возвращает целое число, представляющее общее число элементов во всех измерениях массива.

`Max()` Возвращает максимальное значение, содержащееся в массиве.

`Min()` Возвращает минимальное значение, содержащееся в массиве.

`Reverse()` Изменяет порядок элементов массива на противоположный.

`Sum()` Вычисляет сумму последовательности числовых значений.

СПИСОК ЛИТЕРАТУРЫ

1. Есипов А.С., Паньгина Н.Н., Громада М.И. Информатика: сборник задач и решений для общеобразовательных учебных заведений. – СПб.: Наука и техника, 2001. – 368 с.
2. Окулов С.М. Программирование в алгоритмах. – М.: Бином; Лаборатория знаний, 2004. – 341 с.
3. Юркин А.Г. Задачник по программированию. – СПб.: Питер, 2002. – 192 с.
4. Троелсен Э. Язык программирования C# 5.0 и платформа .NET 4.5. – М.: Вильямс, 2013. – 1312 с.
5. Албахари Дж. C# 3.0. Справочник: пер. с англ. / Дж. Албахари, Б. Албахари. – 3-е изд. – Спб.: БХВ-Петербург, 2009. – 944 с.: ил.
6. Биллиг В. Основы программирования на C# // Интуит [2013]. Дата обновления: 22.11.2005. URL: http://www.intuit.ru/studies/courses/2247/18/info (дата обращения: 27.06.2013).
7. Павлоская Т. Программирование на языке высокого уровня C# // Интуит [2013]. Дата обновления: 15.09.2010. URL: http://www.intuit.ru/studies/courses/2247/18/info (дата обращения: 27.06.2013).
8. Вихтенко Э.М. Геометрические задачи в олимпиадах по программированию. – Изд-во МИФ-2. – № 2. – 2005.
9. Липский В. Комбинаторика для программистов. – М.: Мир, 1988. – 200 с.
10. Демин А.Ю., Дорофеев В.А. Программирование на C#: учебное пособие. − Томск: Изд-во Томского политехнического университета, 2013. – 134 с.

Учебное издание

ДЁМИН Антон Юрьевич
ДОРОФЕЕВ Вадим Анатольевич

ЛАБОРАТОРНЫЙ ПРАКТИКУМ ПО ИНФОРМАТИКЕ

Учебное пособие

Корректура *В.Ю. Пановица*
Компьютерная верстка *К.С. Чечельницкая*
Дизайн обложки *Т.В. Буланова*

Подписано к печати 19.03.2014. Формат 60x84/16. Бумага «Снегурочка».
Печать XEROX. Усл. печ. л. 7,68. Уч.-изд. л. 6,94.
Заказ 183-14. Тираж 100 экз.

Национальный исследовательский Томский политехнический университет
Система менеджмента качества
Издательства Томского политехнического университета
сертифицирована в соответствии с требованиями ISO 9001:2008

ИЗДАТЕЛЬСТВО ТПУ. 634050, г. Томск, пр. Ленина, 30
Тел./факс: 8(3822)56-35-35, www.tpu.ru