ПРИЛОЖЕНИЕ 2

ФЕДЕРАЛЬНОЕ ГОСУДАРСТВЕННОЕ БЮДЖЕТНОЕ ОБРАЗОВАТЕЛЬНОЕ УЧРЕЖДЕНИЕ ВЫСШЕГО ОБРАЗОВАНИЯ
«БАШКИРСКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ»

Стерлитамакский филиал

Колледж

Фонд оценочных средств

по дисциплине ***ОПЦ.13 Программирование в среде Visual Studio***

Общепрофессиональный цикл, вариативная часть
цикл дисциплины и его часть (обязательная, вариативная)

специальность

09.02.07 ***Информационные системы и программирование***
код наименование специальности

квалификация
Специалист по информационным системам

Разработчик (составитель)

Абрамова В. А.
ученая степень, ученое звание,
категория, Ф.И.О.

Стерлитамак 2024

I Паспорт фондов оценочных средств

1. Область применения

Фонд оценочных средств (ФОС) предназначен для проверки результатов освоения дисциплины «Программирование в среде Visual Studio», входящей в состав программы подготовки специалистов среднего звена по специальности 09.02.07 Информационные системы и программирование. **Работа обучающихся во взаимодействии с преподавателем** 152 часа, на самостоятельную работу 56 часов.

2. Объекты оценивания – результаты освоения дисциплины

ФОС позволяет оценить следующие результаты освоения дисциплины в соответствии с ФГОС специальности 09.02.07 Информационные системы и программирование и рабочей программой дисциплины «Программирование в среде Visual Studio»:

умения:

- определять задачи для поиска информации;
- определять необходимые источники информации;
- планировать процесс поиска;
- оформлять результаты поиска;
- применять средства информационных технологий для решения профессиональных задач; использовать современное программное обеспечение;
- анализировать проектную и техническую документацию;
- оценивать размер минимального набора тестов;
- разрабатывать тестовые пакеты и тестовые сценарии;
- выявлять ошибки в системных компонентах на основе спецификаций;
- организовывать заданную интеграцию модулей в программные средства на базе имеющейся архитектуры и автоматизации бизнес-процессов;
- выполнять тестирование интеграции;
- обучающийся должен уметь применять информационные технологии для выполнения откладки программного модуля в предметных областях.

знания:

- приемы структурирования информации;
- формат оформления результатов поиска информации
- современные средства и устройства информатизации;
- порядок их применения и программное обеспечение в профессиональной деятельности
- модели процесса разработки программного обеспечения;
- основные принципы процесса разработки программного обеспечения;
- основные подходы к интегрированию программных модулей;
- современные технологии и инструменты интеграции;
- модели процесса разработки программного обеспечения;
- основные принципы процесса разработки программного обеспечения;
- основные методы отладки;
- методы и схемы обработки исключительных ситуаций;
- обучающий должен знать навыки для применения ИТ при решении задач предметной области.

Вышеперечисленные умения, знания направлены на формирование у обучающихся следующих **общих и профессиональных компетенций**:

ОК.02. Использовать современные средства поиска, анализа и интерпретации информации и информационные технологии для выполнения задач профессиональной деятельности

ОК.09. Пользоваться профессиональной документацией на государственном и иностранном языках

ПК.2.1. Разрабатывать требования к программным модулям на основе анализа проектной и технической документации на предмет взаимодействия компонент.

ПК.2.2. Выполнять интеграцию модулей в программное обеспечение.

ПК.2.3. Выполнять отладку программного модуля с использованием специализированных программных средств.

ПК.2.4. Осуществлять разработку тестовых наборов и тестовых сценариев для программного обеспечения.

ПК.2.5. Производить инспектирование компонент программного обеспечения на предмет соответствия стандартам кодирования.

3 Формы контроля и оценки результатов освоения дисциплины

Контроль и оценка результатов освоения – это выявление, измерение и оценивание знаний, умений и формирующихся общих и профессиональных компетенций в рамках освоения дисциплины.

В соответствии с учебным планом специальности 09.02.07 «Информационные системы и программирование», рабочей программой дисциплины ОПЦ.13. Программирование в среде Visual Studio предусматривается текущий и промежуточный контроль результатов освоения.

3.1 Формы текущего контроля

Текущий контроль успеваемости представляет собой проверку усвоения учебного материала, регулярно осуществляемую на протяжении курса обучения.

Текущий контроль результатов освоения дисциплины в соответствии с рабочей программой и календарно-тематическим планом происходит при использовании следующих обязательных форм контроля:

– выполнение и защита практических работ,

– проверка выполнения самостоятельной работы студентов,

– проверка выполнения контрольных работ,

Во время проведения учебных занятий дополнительно используются следующие формы текущего контроля – *устный опрос, решение задач, тестирование по темам отдельных занятий.*

Выполнение и защита практических работ. Практические работы проводятся с целью усвоения и закрепления практических умений и знаний, овладения профессиональными компетенциями. В ходе практической работы студенты приобретают умения, предусмотренные рабочей программой дисциплины, учатся использовать формулы, и применять различные методики расчета, анализировать полученные результаты и делать выводы, опираясь на теоретические знания.

Список практических работ:

- Практическая работа №1. Работа с решением. Создание проекта.
- Практическая работа №2. Создание консольных приложений.
- Практическая работа №3. Решение задач с помощью циклов и ветвлений.
- Практическая работа №4. Литералы.
- Практическая работа №5. Кортежи.
- Практическая работа №6. Создание классов.
- Практическая работа №7. Операции: арифметические, логические. Операции над

строковыми и символьными данными.

- Практическая работа №8. Ссылочные типы данных.
- Практическая работа №9. Класс Random. Методы и свойства класса System.Array.
- Практическая работа №10. Перегрузка методов. Рекурсивные методы.
- Практическая работа №11. Создание приложения с консольным вводом-выводом.
- Практическая работа №12. Асинхронный ввод-вывод. Потоки символов.
- Практическая работа №13. Работа с каталогами и файлами. Сохранение объектов.
- Практическая работа №14. Работа с бинарными файлами.
- Практическая работа №15. Создание и чтение сжатых файлов.
- Практическая работа №16. Создание классов.
- Практическая работа №17. Программирование классов.
- Практическая работа №18. Инкапсуляция.
- Практическая работа №19. Наследование и полиморфизм.
- Практическая работа №20. Абстрактные классы.
- Практическая работа №21. Реализация интерфейса.
- Практическая работа №22. Многопоточные приложения. Делегаты и события.
- Практическая работа №23. Класс Object.
- Практическая работа №24. Создание программы в среде Visual Studio.
- Практическая работа №25. Элемент «Форма». Свойства формы.
- Практическая работа №26. Класс Control. Элементы управления.
- Практическая работа №27. Элементы управления: группа элементов управления.
- Практическая работа №28. Элементы управления: метка, метка-ссылка.
- Практическая работа №29. Кнопки и двоичные переключатели: кнопка.
- Практическая работа №30. Кнопки и двоичные переключатели: флажок, переключатель.
- Практическая работа №31. Элемент управления с поддержкой редактирования текста: текстовое поле.
- Практическая работа №32. Элемент управления с поддержкой редактирования текста: поле ввода с форматированием.
- Практическая работа №33. Списки изображений и всплывающие подсказки.
- Практическая работа №34. Поле со списком.
- Практическая работа №35. Деревья: древовидное представление TreeView.
- Практическая работа №36. Списковое представление ListView.
- Практическая работа №37. Работа с компонентами меню.
- Практическая работа №38. Работа с панелями инструментов.
- Практическая работа №39. Работа с диалоговыми окнами. Сохранение и открытие файла.
- Практическая работа №40. Работа с диалоговыми окнами. Установка шрифта, цвета.
- Практическая работа №41. Изучение свойств проекта.
- Практическая работа №42. Использование таймера.
- Практическая работа №43. Работа с датами и временем.
- Практическая работа №44. Сборка мусора, управление памятью и указатели.
- Практическая работа №45. Основы LINQ.
- Практическая работа №46. Сериализация.
- Практическая работа №47. Построение диаграмм. Компонент Chart.

- Практическая работа №48. Работа с графикой.
- Практическая работа №49. Использование технологии OLE для связывания и внедрения объектов в различные документы.
- Практическая работа №50. Разработка и использование элементов ActiveX.
- Практическая работа №51. Создание и использование динамически подключаемой библиотеки на языке C#.
- Практическая работа №52. Интеграция и использование динамически подключаемой библиотеки в клиентском приложении на языке C#.

ПРАКТИЧЕСКАЯ РАБОТА 1. РАБОТА С РЕШЕНИЕМ. СОЗДАНИЕ ПРОЕКТА

Изучение основ применения Visual StudioЦель занятия

Получить практические навыки создания проектов с использованиеVisualStudio

Перечень оборудования и программного обеспечения

Персональный компьютер Microsoft Office (Word, Visio)Microsoft Visual Studio 2015

Краткие теоретические сведенияVisualStudio

Интегрированная среда разработки (IntegratedDevelopmentEnvironment — IDE) VisualStudio 2010 предлагает ряд высокоуровневых функциональных возможностей, которые выходят за рамки базового управления кодом.

Встроенный Web-сервер. Для обслуживания Web-приложения ASP.NET необходим Web-сервер, подобный IIS, который будет ожидать Web-запросы и обрабатывать соответствующие страницы. Установить свойWeb-сервер несложно, но может быть не совсем удобно. Наличие в VisualStudio интегрированного Web-сервера позволяет запускать Web-сайтпрямо из среды проектирования, а также повышает безопасность.

Поддержка множества языков при разработке. VisualStudio позволяет писать код на своем языке или любых других предпочитаемых языках, используя все время один и тот же интерфейс (IDE).

Меньше кода для написания. Для создания большинства приложений требуется достаточное количество стандартного стереотипного кода. В VisualStudio такой код генерируется автоматически.

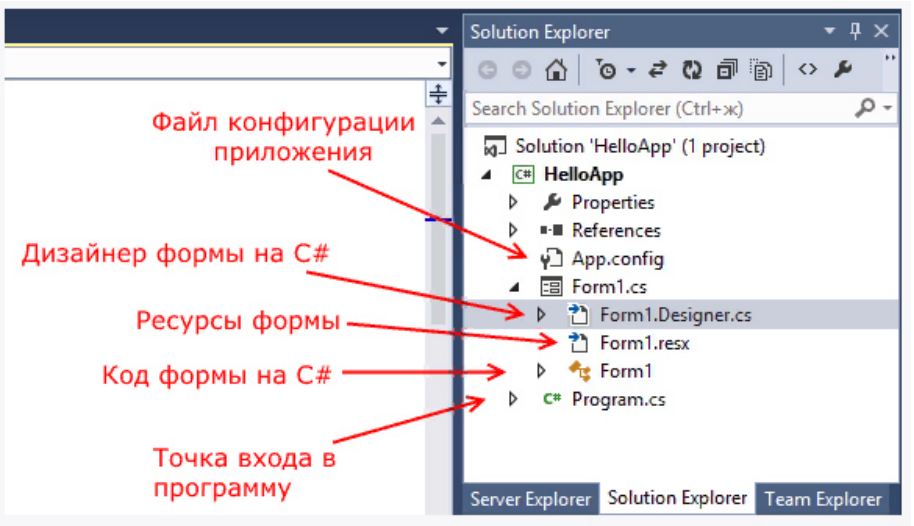
Интуитивный стиль кодирования. По умолчанию VisualStudio форматирует код по мере его ввода, автоматически вставляя необходимые отступы и применяя цветовое кодирование для выделения элементов типа комментариев. Такие незначительные отличия делают код более удобным для чтения и менее подверженным ошибкам. Многие из функциональных возможностей VisualStudio направлены на то, чтобы помогать разработчику делать свою работу как можно быстрее. Удобные функции, вроде функции IntelliSense (которая умеет перехватывать ошибки и предлагать правильные варианты), функции поиска и замены (которая позволяет отыскивать ключевые слова как в одном файле, так и во всем проекте) и функции автоматического добавления и удаления комментариев (которая может
временно скрывать блоки кода), позволяют разработчику работать быстро иэффективно.

Возможности отладки. Предлагаемые в VisualStudio инструменты отладки являются наилучшим средством для отслеживания загадочных ошибок и диагностирования странного поведения. Разработчик может выполнять свой код по строке за раз, устанавливать интеллектуальные точки прерывания, при желании сохраняя их для использования в будущем, и в любое время просматривать текущую информацию из памяти.

Microsoft .NET Framework

.NET Framework – среда для создания и запуска приложений. Все приложения не просто используют библиотеки .NET Framework, но и выполняются под управлением ее компонент.

Её основные компоненты:
общеязыковая исполняющая среда (CommonLanguageRuntime) – динамический компонент;
библиотека классов .NET FrameworkClassLibrary – статический компонент.
Пространство имен
.NET Framework располагает большим набором полезных функций. Каждая из них является членом какого-либо класса. Классы группируются по пространствам имен. Это означает, что в общем случае имя класса может иметь сложную структуру — состоять из последовательности имен, разделенных между собой точками. Последнее имя в этой последовательности собственно и является именем класса. Классы, имена которых различаются лишь последними членами (собственно именами классов) последовательностей, считаются принадлежащими одному пространству имен.
Помещение типа в пространство имен присваивает этому типу длинное имя, состоящее из всех пространств, как серии их имен, разделенных точками (.) и заканчивающееся именем класса.
Если не использовать оператор using, для корректного обращения к функциям необходимо писать полный путь класса, что является достаточноемкой работой.
Средством "навигации" по пространствам имен, а точнее, средством, которое позволяет сокращать имена классов, является оператор
using<ИмяПространстваИмен>;
В приложении может объявляться собственное пространство имен, а также могут использоваться ранее объявленные пространства.
Оператор using сам по себе не обеспечивает доступа к именам, находящимся в других пространствах имен. До тех пор, пока код из пространства имен не будет каким-либо способом привязан к нашему проекту (например, описан в исходном файле проекта или описан в каком- либо коде), привязанному к этому проекту, мы не получим доступа к содержащимся в нем именам.
Основы форм
Внешний вид приложения является нам преимущественно через формы. Формы являются основными строительными блоками. Они предоставляют контейнер для различных элементов управления. А механизм событий позволяет элементам формы отзываться на ввод пользователя, и, таким образом, взаимодействовать с пользователем.
При открытии проекта в Visual Studio в графическом редакторе мы можем увидеть визуальную часть формы - ту часть, которую мы видим после запуска приложения и куда мы переносим элементы с панели управления. Но на самом деле форма скрывает мощный функционал, состоящий из методов, свойств, событий и прочее. Рассмотрим основные свойства форм.
Если мы запустим приложение, то нам отобразится одна пустая форма.
Однако даже такой простой проект с пустой формой имеет несколько компонентов:



Несмотря на то, что мы видим только форму, но стартовой точкой входа в графическое приложение является класс Program, расположенный вфайле Program.cs:

```
using System;
using System.Collections.Generic;using System.Linq;
using System.Threading.Tasks;using System.Windows.Forms;

namespace HelloApp
{
static  class Program
{
[STAThread] static void Main()
{
Application.EnableVisualStyles();
Application.SetCompatibleTextRenderingDefault(false);Application.Run(new Form1());
}
}
}
```

Сначала программой запускается данный класс, затем с помощью выражения Application.Run(new Form1()) он запускает форму Form1. Если вдруг мы захотим изменить стартовую форму в приложении на какую- нибудь другую, то нам надо изменить в этом выражении Form1 на соответствующий класс формы.
Сама форма сложна по содержанию. Она делится на ряд компонентов. Так, в структуре проекта есть файл Form1.Designer.cs, который выглядит примерно так:

```
namespace HelloApp
{
partial class Form1
{
/// <summary>
/// Required designer variable.
/// </summary>
private System.ComponentModel.IContainer components = null;

/// <summary>
/// Clean up any resources being used.
/// </summary>
/// <param name="disposing">true if managed resources should be disposed; otherwise, false.</param>
protected override void Dispose(bool disposing)
{
if (disposing && (components != null))
{
components.Dispose();
}
base.Dispose(disposing);
```

```
}

#region Windows Form Designer generated code

/// <summary>
/// Required method for Designer support - do not modify
/// the contents of this method with the code editor.
/// </summary>
private void InitializeComponent()
{
this.SuspendLayout();
//
// Form1
//
this.AutoScaleDimensions = new System.Drawing.SizeF(6F, 13F); this.AutoScaleMode = System.Windows.Forms.AutoScaleMode.Font; this.ClientSize = new System.Drawing.Size(284, 261);
this.Name = "Form1"; this.Text = "Привет мир!";this.ResumeLayout(false);

}

#endregion

}
}
```

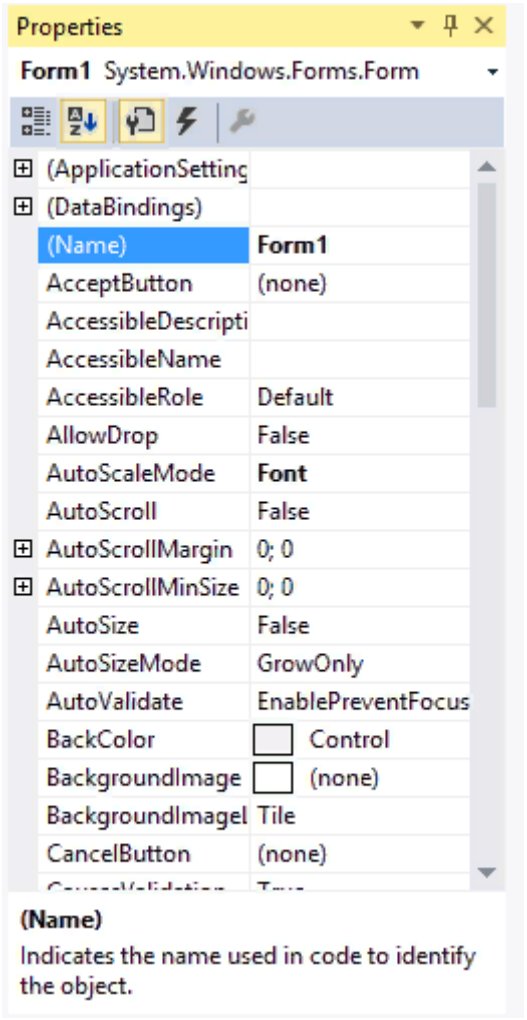
Здесь объявляется частичный класс формы Form1, которая имеет два метода: Dispose(), который выполняет роль деструктора объекта, и InitializeComponent(), который устанавливает начальные значения свойствформы.
При добавлении элементов управления, например, кнопок, их описаниетакже добавляется в этот файл.
Но на практике мы редко будем сталкиваться с этим классом, так как они выполняет в основном дизайнерские функции - установка свойств объектов, установка переменных.
Еще один файл - Form1.resx - хранит ресурсы формы. Как правило, ресурсы используются для создания однообразных форм сразу для нескольких языковых культур.
И более важный файл - Form1.cs, который в структуре проекта называется просто Form1, содержит код или программную логику формы:

```
using System;
using System.Collections.Generic;using System.ComponentModel; using System.Data;
using System.Drawing;using System.Linq; using System.Text;
using System.Threading.Tasks;using System.Windows.Forms;

namespace HelloApp
{
public partial class Form1 : Form
{
public Form1()
```

```
{
InitializeComponent();
}
}
}
```

По умолчанию здесь есть только конструктор формы, в котором просто вызывается метод InitializeComponent(), объявленный в файле дизайнера Form1.Designer.cs. Именно с этим файлом мы и будем больше работать.
С помощью специального окна Properties (Свойства) Visual Studio предоставляет нам удобный интерфейс для управления свойствами элемента:



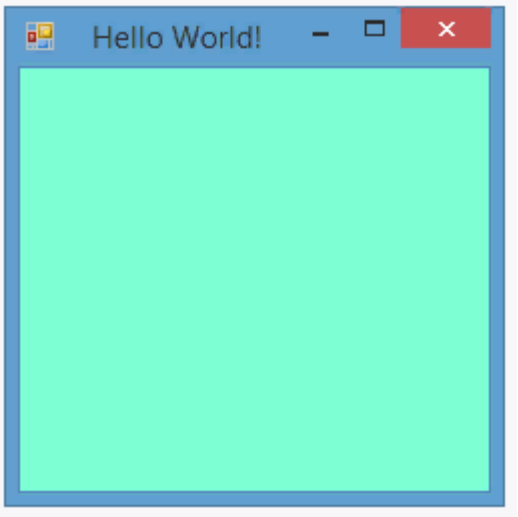
Большинство этих свойств оказывает влияние на визуальное отображение формы. Пробежимся по основным свойствам:
Name: устанавливает имя формы - точнее имя класса, который наследуется от класса Form
BackColor: указывает на фоновый цвет формы. Щелкнув на это свойство, мы сможем выбрать тот цвет, который нам подходит из списка предложенных цветов или цветовой палитры
BackgroundImage: указывает на фоновое изображение формы

BackgroundImageLayout: определяет, как изображение, заданное в свойстве BackgroundImage, будет располагаться на форме.
ControlBox: указывает, отображается ли меню формы. В данном случае под меню понимается меню самого верхнего уровня, где находятся иконка приложения, заголовок формы, а также кнопки минимизации формы и крестик. Если данное свойство имеет значение false, то мы не увидим ни иконку, ни крестика, с помощью которого обычно закрывается форма
Cursor: определяет тип курсора, который используется на форме
Enabled: если данное свойство имеет значение false, то она не сможет получать ввод от пользователя, то есть мы не сможем нажать на кнопки, ввести текст в текстовые поля и т.д.
Font: задает шрифт для всей формы и всех помещенных на нее элементов управления. Однако, задав у элементов формы свой шрифт, мы можем тем самым переопределить его
ForeColor: цвет шрифта на форме
FormBorderStyle: указывает, как будет отображаться граница формы и строка заголовка. Устанавливая данное свойство в None можно создавать внешний вид приложения произвольной формы
HelpButton: указывает, отображается ли кнопка справки формы
Icon: задает иконку формы
Location: определяет положение по отношению к верхнему левому углу экрана, если для свойства StartPosition установлено значение Manual
MaximizeBox: указывает, будет ли доступна кнопка максимизации окна в заголовке формы
MinimizeBox: указывает, будет ли доступна кнопка минимизации окна
MaximumSize: задает максимальный размер формы **MinimumSize:** задает минимальный размер формы **Opacity:** задает прозрачность формы
Size: определяет начальный размер формы
StartPosition: указывает на начальную позицию, с которой форма появляется на экране
Text: определяет заголовок формы
TopMost: если данное свойство имеет значение true, то форма всегда будет находиться поверх других окон
Visible: видима ли форма, если мы хотим скрыть форму отпользователя, то можем задать данному свойству значение false
WindowState: указывает, в каком состоянии форма будет находиться при запуске: в нормальном, максимизированном или минимизированном
Программная настройка свойств
С помощью значений свойств в окне Свойства мы можем изменить посвоему усмотрению внешний вид формы, но все то же самое мы можем сделать динамически в коде. Перейдем к коду, для этого нажмем правой кнопкой мыши на форме и выберем в появившемся контекстном меню ViewCode (Просмотр кода). Перед нами открывается файл кода Form1.cs. Изменим его следующим образом:

```
using System;
using System.Collections.Generic;using System.ComponentModel; using System.Data;
using System.Drawing;using System.Linq; using System.Text;
using System.Threading.Tasks;using System.Windows.Forms;

namespace HelloApp
```

```
{
public partial class Form1 : Form
{
public Form1()
{
InitializeComponent();Text = "Hello World!";
this.BackColor = Color.Aquamarine;this.Width = 250;
this.Height = 250;
}
}
}
```



В данном случае мы настроили несколько свойств отображения формы: заголовок, фоновый цвет, ширину и высоту. При использовании конструктора формы надо учитывать, что весь остальной код должен идти после вызова метода InitializeComponent(), поэтому все установки свойств здесь расположены после этого метода.

Установка размеров формы

Для установки размеров формы можно использовать такие свойства как Width/Height или Size. Width/Height принимают числовые значения, как в вышеприведенном примере. При установке размеров через свойство Size, нам надо присвоить свойству объект типа Size:

```
this.Size = new Size(200,150);
```

Объект Size в свою очередь принимает в конструкторе числовые значения для установки ширины и высоты.

Начальное расположение формы

Начальное расположение формы устанавливается с помощью свойства StartPosition, которое может принимать одно из следующих значений:

Manual: Положение формы определяется свойством Location **CenterScreen:** Положение формы в центре экрана **WindowsDefaultLocation:** Позиция формы на экране задается системой

Windows, а размер определяется свойством Size

WindowsDefaultBounds: Начальная позиция и размер формы на экране задается системой Windows
CenterParent: Положение формы устанавливается в центре родительского окна
Все эти значения содержатся в перечислении FormStartPosition, поэтому, чтобы, например, установить форму в центре экрана, нам надо прописать так:

```
this.StartPosition = FormStartPosition.CenterScreen;
```

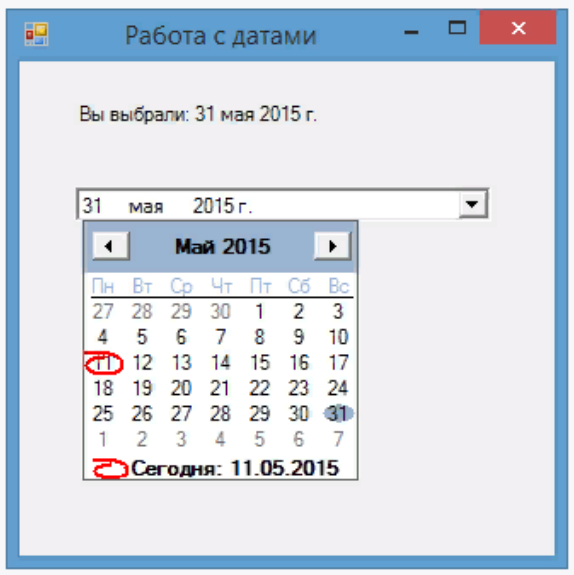
Фон и цвета формы
Чтобы установить цвет как фона формы, так и шрифта, нам надо использовать цветовое значение, хранящееся в структуре Color:

```
this.BackColor = Color.Aquamarine;this.ForeColor = Color.Red;
```

Кроме того, мы можем в качестве фона задать изображение в свойстве BackgroundImage, выбрав его в окне свойств или в коде, указав путьк изображению:
this.BackgroundImage = Image.FromFile("C:\\Users\\Eugene\\Pictures\\3332.jpg");

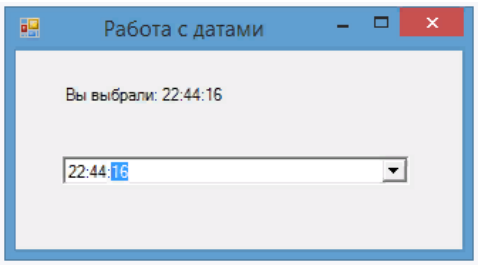
Чтобы должным образом настроить нужное нам отображение фоновой картинки, надо использовать свойство BackgroundImageLayout, которое может принимать одно из следующих значений:
None: Изображение помещается в верхнем левом углу формы и сохраняет свои первоначальные значения
Tile: Изображение располагается на форме в виде мозаики
Center: Изображение располагается по центру формы
Stretch: Изображение растягивается до размеров формы без сохранения пропорций
Zoom: Изображение растягивается до размеров формы с сохранениемпропорций
Например, расположим форму по центру экрана: this.StartPosition = FormStartPosition.CenterScreen;
Для работы с датами в Windows Forms имеются элементы DateTimePicker и MonthCalendar.

DateTimePicker
DateTimePicker представляет раскрывающийся по нажатию календарь, в котором можно выбрать дату. собой элемент, который с помощью перемещения ползунка позволяет вводить числовые значения.

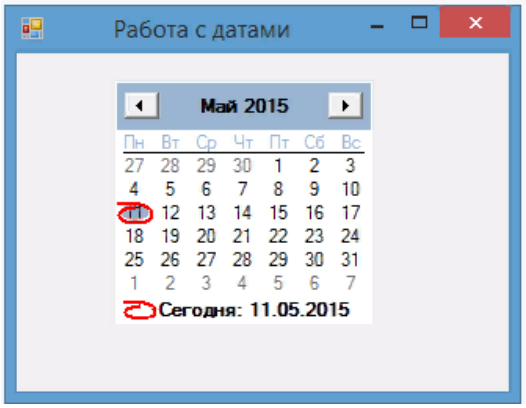


Наиболее важные свойства DateTimePicker:
Format: определяет формат отображения даты в элементе управления.
Может принимать следующие значения: **Custom:** формат задается разработчиком **Long:** полная дата
Short: дата в сокращенном формате
Time: формат для работы с временем
CustomFormat: задает формат отображения даты, если для свойства Format установлено значение Custom
MinDate: минимальная дата, которую можно выбрать
MaxDate: наибольшая дата, которую можно выбрать
Value: определеяте текущее выбранное значение в DateTimePicker
Text: представляет тот текст, который отображается в элементе

Свойство Value хранит объект DateTime, поэтому с ним можно работать как и с любой другой датой. В данном случае выбранная дата преобразуется в строку времени.



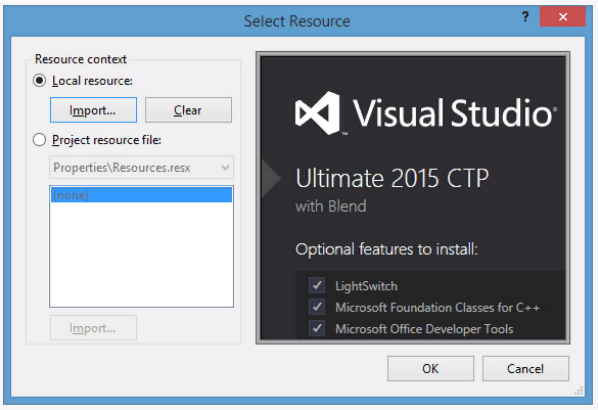
MonthCalendar
С помощью MonthCalendar также можно выбрать дату, только в данном случае этот элемент представляет сам календарь, который не надо раскрывать:



Рассмотрим некоторые основные свойства элемента.
Свойства выделения дат:
AnnuallyBoldedDates: содержит набор дат, которые будут отмеченыжирным в календаре для каждого года
BoldedDates: содержит набор дат, которые будут отмечены жирным(только для текущего года)
MonthlyBoldedDates: содержит набор дат, которые будут отмеченыжирным для каждого месяца

Свойства для определения дат в календаре:
MinDate: определяет минимальную дату для выбора в календаре **MaxDate:** задает наибольшую дату для выбора в календаре **FirstDayOfWeek:** определяет день недели, с которого должна начинаться неделя в календаре
SelectionRange: определяет диапазон выделенных дат **SelectionEnd:** задает конечную дату выделения **SelectionStart:** определяет начальную дату выделения
ShowToday: при значении true отображает внизу календаря текущую дату
ShowTodayCircle: при значении true текущая дата будет обведена кружочком
TodayDate: определяет текущую дату. По умолчанию используется системная дата на компьютере, но с помощью данного свойства мы можем ее изменить.

Элемент PictureBox
PictureBox предназначен для показа изображений. Он позволяет отобразить файлы в формате bmp, jpg, gif, а также метафайлы ищображений и иконки. Для установки изображения в PictureBox можно использовать рядсвойств:
Image: устанавливает объект типа Image
ImageLocation: устанавливает путь к изображению на диске или в интернете
InitialImage: некоторое начальное изображение, которое будет отображаться во время загрузки главного изображения, которое хранится всвойстве Image
ErrorImage: изображение, которое отображается, если основное изображение не удалось загрузить в PictureBox
Чтобы установить изображение в Visual Studio, надо в панели СвойствPictureBox выбрать свойство Image. В этом случае нам откроется окно импорта изображения в проект, где мы собственно и сможем выбрать нужное изображение на компьютере и установить его для PictureBox:



И затем мы сможем увидеть данное изображение в PictureBox:

Либо можно загрузить изображение в коде:

pictureBox1.Image = Image.FromFile("C:\Users\Eugene\Pictures\12.jpg");

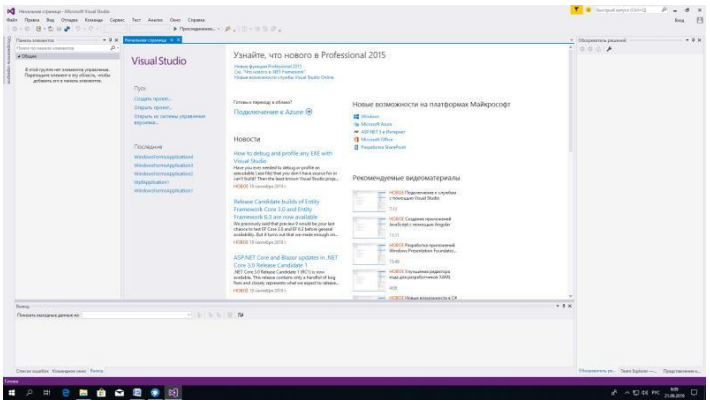
Размер изображения
Для установки изображения в PictureBox используется свойство SizeMode, которое принимает следующие значения:
Normal: изображение позиционируется в левом верхнем углу PictureBox, и размер изображения не изменяется. Если PictureBox больше размеров изображения, то по справа и снизу появляются пустоты, если меньше - то изображение обрезается
StretchImage: изображение растягивается или сжимается таким обраом, чобы вместиться по всей ширине и высоте элемента PictureBox
AutoSize: элемент PictureBox автоматически растягивается, подстраиваясь под размеры изображения
CenterImage: если PictureBox меньше изображения, то изображение обрезается по краям и выводится только его центральная часть. Если же PictureBox больше изображения, то оно позиционируется по центру.
Zoom: изораженияе подстраивается под размеры PictureBox, сохраняяпри этом пропорции

Задания

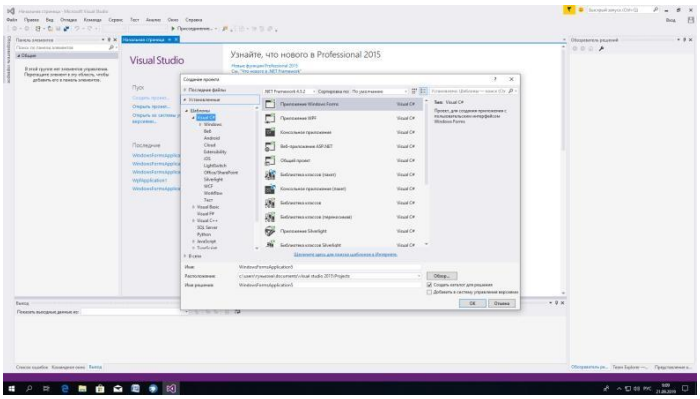
Изучить теоретические сведения.
Создать сообщение в консольном приложении.
Создать приложение WindowsForms по варианту, в которое добавить кнопку ВЫХОД.
Порядок выполнения работы

Задание 1 Изучение окон и служб среды разработки VisualStudio

После вызова **VisualStudio** на экране появляется главное окно.
Основными элементами являются:
строка заголовка, содержащая название программы и имя открытого файла, а также кнопки свертывания, восстановления и закрытия;
строка меню, которая представляет доступ ко всем функциям и командам;
панель элементов, состоящая из нескольких закладок, на которых располагаются визуальные компоненты, используемые при создании программ;
окно свойств, в котором производится настройка основных свойств визуальных компонентов;
список ошибок, содержащий ошибки, найденные транслятором на всех этапах построения проекта:
обозреватель решений обеспечивает выбор решения, проекта, исходного файла, ссылок на внешние сборки и прочих ресурсов, которые иобразуют проект.



В меню Файл последовательно выберите пункты Создать и Проект.



В области Категории шаблонов откройте Visual C#, а затем щелкнитеWindows.
В области Шаблоны щелкните Консольное приложение. Введите имя проекта в поле Имя. Нажмите кнопку ОК. В Обозревателе решений появится новый проект.
Если файл Program.cs не открыт в редакторе кода, щелкните правой кнопкой мыши Program.cs в обозревателе решений, а затем нажмите кнопкуПросмотреть код.
Любая программа на языке C# - это набор классов, которые взаимодействуют друг с другом.
В одном из классов программы должна находиться, так называя «точка входа» - статический метод Main. Наличие или отсутствие этого метода определяет тип получаемого результата компиляции – сборки.
Если метод присутствует – получаем исполняемую программу EXE, впротивном случае – библиотеку DLL. Классы могут быть вложены друг в друга. Но точка входа должна быть только в одном.
Класс определяется с помощью ключевого слова class, после которогоидет имя класса. Тело класса заключается в фигурные скобки.
Язык программирования C# чувствителен к регистру символов, поэтому метод Main должен начинаться с большой буквы. Кроме того, этотметод должен быть определен как static, что позволит вызвать метод без создания объекта класса. Заголовок можно безболезненно упростить, удалив аргументы, которые, как правило, не задаются. Они имеют смысл, когда проект вызывается из командной строки, позволяя с помощью параметров задать нужную стратегию выполнения проекта.

Вставьте следующий код в метод Main между фигурнымискобками:

```
Console.WriteLine("Всем привет! Я, <Фамилия, Имя>, здесь и радвстрече!");
Console.ReadKey();
```

Нажмите клавишу F5 для запуска проекта. Появляется окно командностроки, содержащее строку.

```
16
```

2 Создание приложения WindowsForms

Ядром Windows-программ, написанных на C#, является форма. Форма инкапсулирует основные функции, необходимые для: 1) создания окна, 2) его отображения на экране и 3) получения сообщений. Форма может представлять собой окно любого типа, включая основное окно приложения,дочернее или даже диалоговое окно.

Первоначально окно создается пустым. Затем в него добавляются меню и элементы управления, например экранные кнопки, списки и флажки. Таким образом, форму можно представить в виде контейнера для других Windows-объектов.

Событие — это нечто, инициируемое действиями пользователя при работе с оконным интерфейсом.

В Windows определены различные типы управляющих элементов: экранные кнопки, флажки, переключатели, окна списков.

Несмотря на различия между ними, способы их создания и обработки примерно одинаковы.

В меню Файл выберите команду Создать и щелкните Проект. Выберите шаблон Приложение WindowsForms, в поле Имя введите MyProject и нажмите кнопку ОК.

Откроется конструктор WindowsForms с формой Windows. Это пользовательский интерфейс для создаваемого приложения.

Несмотря на то, что мы видим только форму, но стартовой точкой входа в графическое приложение является класс Program, расположенный вфайле Program.cs:

В меню Вид щелкните Панель элементов, чтобы открыть список элементов управления.

Разверните список Стандартные элементы управления и перетащите элемент управления Label.

Также, из списка панели элементов Стандартные элементы управления перетащите в форму кнопку Button и поместите ее рядом с подписью.

Дважды щелкните новую кнопку, чтобы открыть редактор кода. Visual C# вставил метод с именем button1_Click, который выполняется при нажатии кнопки.

Изменитеметодследующимобразом.

```
private void button1_Click(object sender, EventArgs e)
{
label1.Text = "Hello, World!";
}
```

Нажмите клавишу F5 чтобы скомпилировать и запустить приложение.

При нажатии на кнопку теперь будет выводиться текстовое сообщение.Элемент управления TextBox позволяет осуществлять ввод текстовой информации. Введенное значение помещается в textBox.Text

Для преобразования переменных в другой тип воспользуйтесь методомConvert.

Для выдачи предупреждающих сообщений используйте метод MessageBox.Show(“Сообщение”)

Для выхода из проекта используйте метод Application.Exit(). Значения progressBar и trackBar хранятся в свойстве Value.

Чтобы выполнять действия с элементами формы, содержащими числовые значения, необходимо преобразовать их в числовой формат, т.к. на форме все отображается в текстовом виде. Для этого необходимо задать команду преобразования Convert.ToInt32(<текстовое значение>).
Например, Convert.ToInt32(label1.Text).
Чтобы преобразовать значения вычислений в текстовый формат, используем Convert.ToSnring(<значение>).
Например,

```
textBox1.Text = Convert.ToString(2 + 6 - 1);или
textBox1.Text = (2 + 6 - 1).ToString();
```

Содержание отчета

Название работы
Цель работы
Технические средства обучения
Задания (условия задач)
Порядок выполнения работы
Ответы на контрольные вопросы
Вывод

Варианты заданий
Задание 3
Разработка проекта, содержащего форму, на которую помещены двекнопки и три элемента управления label. При нажатии на первую кнопку в элементе управления label1 выдается ваша фамилия, в label2 – имя, в label3 – отчество, при нажатии на вторую – надписи меняются местами.
Разработка проекта, содержащего форму, на которую помещена кнопка и элементы управления label и textBox. При нажатии на кнопку содержимое элемента управления textBox1 выдается в элементе управленияlabel1.
Разработка проекта, содержащего форму, на которую помещена кнопка и элемент управления label, содержащий числовое значение. При каждом нажатии на кнопку к содержимому элемента управления label1 добавляется еще единица.
Разработка проекта, содержащего форму, на которую помещена кнопка управления, label и 2 элемента textBox. При нажатии на кнопку содержимое элементов управления textBox1 и textBox2 выдается в элементе управления label1.
Разработка проекта, содержащего форму, на которую помещены элементы управления textBox и label с текстом «Было набрано ». При каждом изменении значения textBox его содержимое добавляется в label после слов «, а затем ».
Разработка проекта, содержащего форму, на которую помещены кнопка, элементы управления textBox и label. При нажатии на кнопку содержимое textBox выдается в label.
Разработка проекта, содержащего форму, на которую помещены кнопка, два элемента управления textBox, содержащих числа, и label. При нажатии на кнопку содержимое textBox складывается и выдается после содержимого label в виде сообщения в MessageBox.

Разработка проекта, содержащего форму, на которую помещена кнопка и элемент управления label с произвольным текстом. При нажатии на кнопку содержимое элемента управления label изменяется на название вашей группы.

Разработка проекта, содержащего форму, на которую помещена кнопка и элемент управления label, в котором содержится слово «сегодня» и dateTimePicker. При нажатии на кнопку содержимое элемента управления dateTimePicker добавляется в label.

Разработка проекта, содержащего форму, на которую помещена кнопка и элемент управления textBox. При нажатии на кнопку содержимое элемента управления textBox выдается в виде сообщения в MessageBox.

Разработка проекта, содержащего форму, на которую помещена кнопка и два элемента управления label. При нажатии на кнопку в MessageBox выдается содержимое элементов управления label1 и label2.

Разработка проекта, содержащего форму, на которую помещена кнопка и элементы управления textBox и dateTimePicker. При нажатии на кнопку содержимое элемента управления dateTimePicker выдается в textBox.

Разработка проекта, содержащего форму, на которую помещена кнопка и элементы управления textBox и label, содержащие числовые значение. При каждом нажатии на кнопку к содержимому элемента управления label1 добавляется содержимое элемента управления textBox.

Разработка проекта, содержащего форму, на которую помещена кнопка и элемент управления label, содержащий числовое значение. При каждом нажатии на кнопку к содержимому элемента управления label1 добавляется еще единица.

Разработка проекта, содержащего форму, на которую помещена кнопка, элементы управления label и monthCalendar. При нажатии на кнопку в элементе управления label1 выдается выбранная дата, а в label2 день недели.

Разработка проекта, содержащего форму, на которую помещена кнопка, элементы управления label, в которых помещен текст, и monthCalendar. При нажатии на кнопку в элементе управления label1 добавляется текущая дата, а в label2 день недели.

Разработка проекта, содержащего форму, на которую помещена кнопка, элемент управления pictureBox. При нажатии на кнопку в элементе управления pictureBox добавляется изображение из файла, ссылка на который начинается с @, а затем следует полный путь к файлу в кавычках, например @"C:\Users\...\круг.png"

Разработка проекта, содержащего форму, на которую помещена кнопка, элементы управления label и dateTimePicker1. При нажатии на кнопку в элементе управления label1 выдается выбранная дата, а в label2 деньнедели.

Разработка проекта, содержащего форму, на которую помещена кнопка, элементы управления label, в которых помещен текст, и dateTimePicker1. При нажатии на кнопку в элементе управления label1 добавляется текущая дата, а в label2 день недели.

Разработка проекта, содержащего форму, на которую помещена кнопка и элемент управления textBox. При нажатии на кнопку меняется цвет фона, цвет фона в textBox и цвет текста.

Разработка проекта, содержащего форму, на которую помещена кнопка. При нажатии на кнопку в качестве фона добавляется изображение из файла, ссылка на который начинается с @, а затем следует полный путь к файлу в кавычках, например @"C:\Users\...\круг.png"

Разработка проекта, содержащего форму, на которую помещена кнопки и элементы управления label. При нажатии на одну кнопку меняетсяцвет фона, а на другую – цвет текста в label.

Разработка проекта, содержащего форму, на которую помещена кнопка и элементы управления label, в котором содержится слово «сегодня» и dateTimePicker. При нажатии на кнопку по выбранной дате в dateTimePicker в label1 добавляется день недели, а в label2 текущее время.

Контрольные вопросы

Что такое Visual Studio?
Основные возможности Visual Studio.
Что такое Microsoft.NET Framework и для чего используется?
Для чего служит консольное приложение?
Что такое форма и для чего предназначена?
Как задать название, цвет и другие параметры формы?
Для чего используется элементы управления label, textBox, кнопка?
Для чего используется элементы управления DateTimePicker и MonthCalendar?
Чем отличаются DateTimePicker и MonthCalendar?

ПРАКТИЧЕСКАЯ РАБОТА № 2 СОЗДАНИЕ КОНСОЛЬНЫХ ПРИЛОЖЕНИЙ

Цели:

- введение в консольные приложения C#.

Задачи:

- базовые сведения о программировании в среде .NET;
- ввод и вывод в консоль;
- строки и массивы;
- структуры и классы;
- сведения о системе.

1.1. Консольное приложение Hello

Главная особенность при программировании на C# — сначала созда- ется класс, в котором определен статический метод Main с параметром илибез него. Код программы описывается внутри этого метода и выполнение программы начинается с него. Другая особенность — всё есть объект, и наследуется от System.Object.

Создадим проект консольного приложения C#, название SharpHello.

После создания проекта файл Program.cs содержит примерно следую- щий код:

```
using System;
using System.Collections.Generic;using System.Linq;
using System.Text;

namespace SharpHello {class Program {
    static void Main(string[] args) {
    }
    }
    }
```

Вместо директивы препроцессора #include в C# используется using и далее используемые пространства имен. Основным пространством имен является System. В нем определено много других пространств, которые подключаются по мере необходимости. Visual Studio автоматически вклю- чает некоторые пространства, это не значит, что все они потребуются.

Части кода C# располагаются в пространствах имен, название кото- рых совпадает с названием проекта, но может изменяться программистом.

Весь код располагается внутри метода Main. Сейчас этот метод со- держит только оператор возврата return 0, если функция Main объявлена как int, или ничего не содержит, если функция Main объявлена как void.

Для ввода и вывода в консоль используется класс Console и его мето- ды Write, WriteLine, Read и ReadLine. Методы со словом Line читают или записывают конец строки (символ newline).

Первое приложение начинается с вывода строки Hello, World!:

```
class Program {
static void Main(string[] args) { Console.WriteLine("Hello, World!");
}
}
```

Для проверки запускаем приложение при помощи Ctrl+F5.

Добавим в программу статический метод, полностью совпадающий сметодом Main, за исключением того, что название метода arguments:

```
namespace SharpHello {class Program {
static void Main(string[] args) {arguments(args);
}
static void arguments(string[] args) { Console.WriteLine("Hello, World!");
}
}
}
```

Следующий тест — вывод параметров функции Main. Система фор-мирует строковый массив параметров, который передается функции Main.

Параметры командной строки передаются как массив, размер массива — это его свойство Length. Для перебора элементов массива можно ис-пользовать цикл foreach.

Начнем с простого, обычного цикла for:

```
static void arguments(string[] args) { Console.WriteLine("Hello, World!");
for (int i = 0; i < args.Length; i++) {Console.WriteLine(args[i]);
}
}
```

Чтобы протестировать этот фрагмент кода, нужно запустить прило- жение из командной строки, при помощи, например, FAR, и добавить при этом несколько параметров, например:

HelloWorld first second 456

Здесь в командной строке три параметра. Скомпилированное прило- жение находится в каталоге bin\Debug. Если все сделано правильно, то в консоль будет выведено:

```
Hello, World!first
second456
```

Для тестирования лучше установить параметры командной строки всвойствах проекта, раздел Debug, Command Arguments.

А теперь пробуем выполнить то же при помощи цикла foreach:

```
foreach (string s in args) {Console.WriteLine(s);
}
```

1.2. Чтение и запись в консоль

Метод WriteLine (Write) позволяет выводить и переменные. Для этого в том месте строки вывода, где требуется вывести переменную, ставится блок {*n*}, где *n* — номер переменной в списке, который следует за строкой через запятую. Будем выводить параметры командной строки с указанием их номера:

```
static void arguments(string[] args) { Console.WriteLine("Hello, World!");
for (int i = 0; i < args.Length; i++) {Console.WriteLine(args[i]);
    }
foreach (string s in args) {Console.WriteLine(s);
    }
for (int i = 0; i < args.Length; i++) { Console.WriteLine("Parameter {0} is {1}", i, args[i]);
    }
    }
```

Тестируем эту часть кода. Убеждаемся, что в консоль выводится:

Parameter 0 is first Parameter 1 is secondParameter 2 is 456

Описываем новый статический метод dialog:

```
static void Main(string[] args) {
//arguments(args);dialog();
}
static void dialog() {
}
```

Чтение консоли выполняет метод ReadLine. Сначала в методе dialog запросим имя пользователя:

```
static void dialog() { Console.Write("Как вас называть: ");
}
```

Затем объявляем переменную s типа string и принимаем в нее то, чтовозвратит метод ReadLine. После этого выводим в консоль приветствие:

```
string s;
s = Console.ReadLine(); Console.WriteLine("Привет, {0}!", s);
```

Далее запрашиваем возраст, принимаем его и выводим:

```
Console.Write("Ваш возраст: ");s = Console.ReadLine();
Console.WriteLine("Вам {0} лет (года).", s);
```

Запускаем программу, вводим запрашиваемые значения.Убеждаемся, что значения переменных выводятся.

ПРАКТИЧЕСКАЯ РАБОТА № 3. РЕШЕНИЕ ЗАДАЧ С ПОМОЩЬЮ ЦИКЛОВ И ВЕТВЛЕНИЙ

Условный оператор if

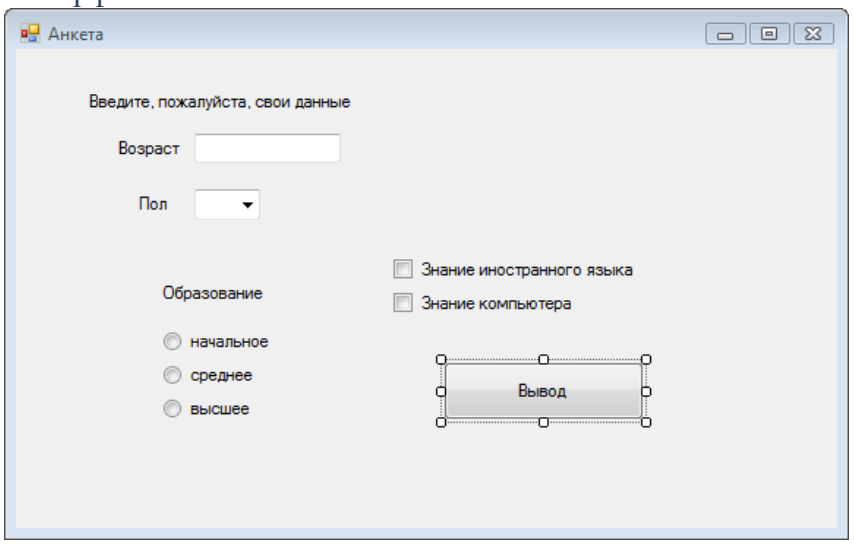
Цель работы: научиться применять условные ветвления при разработке логики программы.

Описание программы

Программа предоставляет пользователю возможность ввести данные о себе в формате анкеты. Для указания пола используется компонент comboBox, образования – radioButton, дополнительных навыков (знания иностранного языка и умения работать за компьютером) – checkBox.

По результатам анкетирования появляется диалоговое окно с сообщением. Претендент получает работу, если: это мужчина в возрасте от 25 до 50 лет с высшим образованием, имеющий навыки работы с компьютером.

Интерфейс



Условный оператор switch

Цель работы: научиться применять оператор множественного выбора.

Описание программы

В старояпонском календаре был принят двенадцатилетний цикл. Годы внутри цикла носили названия животных:

крысы, коровы, тигра, зайца, дракона, змеи, лошади, овцы, обезьяны, петуха, собаки и свиньи.

Написать программу, которая позволяет ввести номер года и печатает его название по старояпонскому календарю.

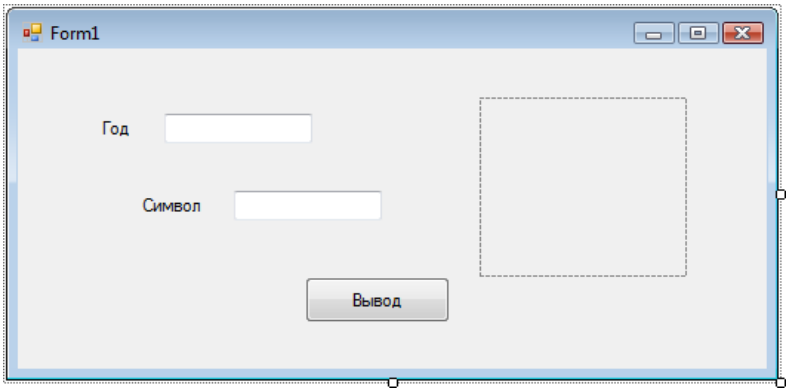
Вводится год, и программа отображает информацию о животном-символе и соответствующую картинку (компонент pictureBox).

Свойство ImageLocation хранит полный путь к файлу картинки.

Чтобы не указывать полный путь, папку с картинками необходимо хранить в том же каталоге, где файл приложения. Для отладки программы из среды C# папку картинок нужно поместить в каталог bin\Debug своего проекта.

Справка: 1996 г. – год крысы – начало очередного цикла.

Интерфейс



Использование цикла в C#

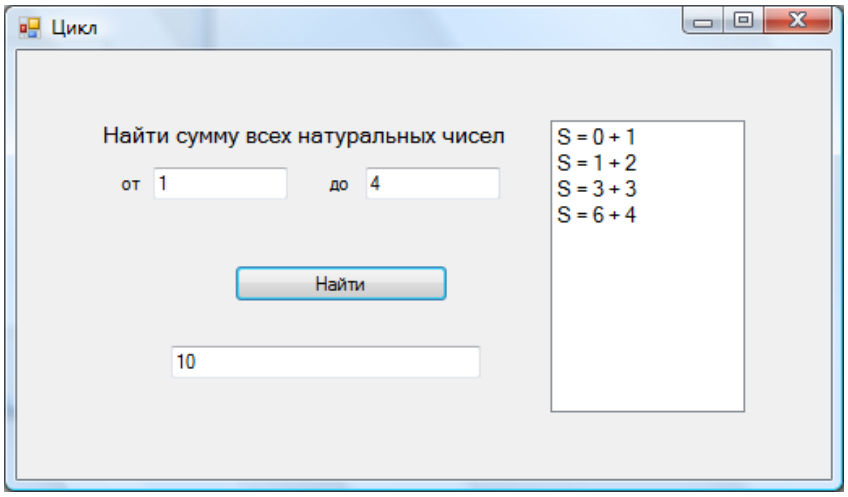
Цель работы: рассмотреть использование классического цикла for.

Описание программы

С помощью цикла программа находит сумму всех натуральных чисел в диапазоне от A до B. A и B вводятся пользователем.

В списке listBox показываются промежуточные результаты суммирования на каждом шаге цикла.

Интерфейс



Цикл for

Цель работы: научиться применять циклы при решении задач.

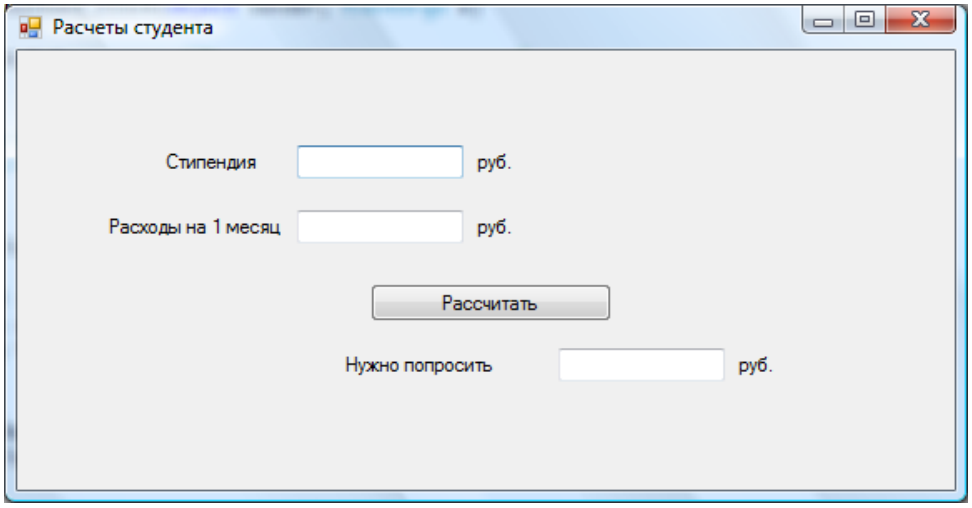
Описание программы

Ежемесячная стипендия студента составляет A руб., а расходы на проживание превышают стипендию и составляют B руб. в месяц.

Рост цен ежемесячно увеличивает расходы на 3%.

Составьте программу расчета суммы денег, которую необходимо единовременно попросить у родителей, чтобы можно было прожить учебный год (10 месяцев), используя только эти деньги и стипендию.

Интерфейс



ПРАКТИЧЕСКАЯ РАБОТА №4. ЛИТЕРАЛЫ

Описание данных

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

В *C#* имеются две общие категории встроенных типовданных: типы значений и ссылочные типы. Они отличаются по содержимому переменной. Концептуально разница между ними состоит в том, что тип значения (*value type*) хранит данные непосредственно, в то время как ссылочный тип (*reference type*) хранит ссылку на значение (рисунок 1).

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

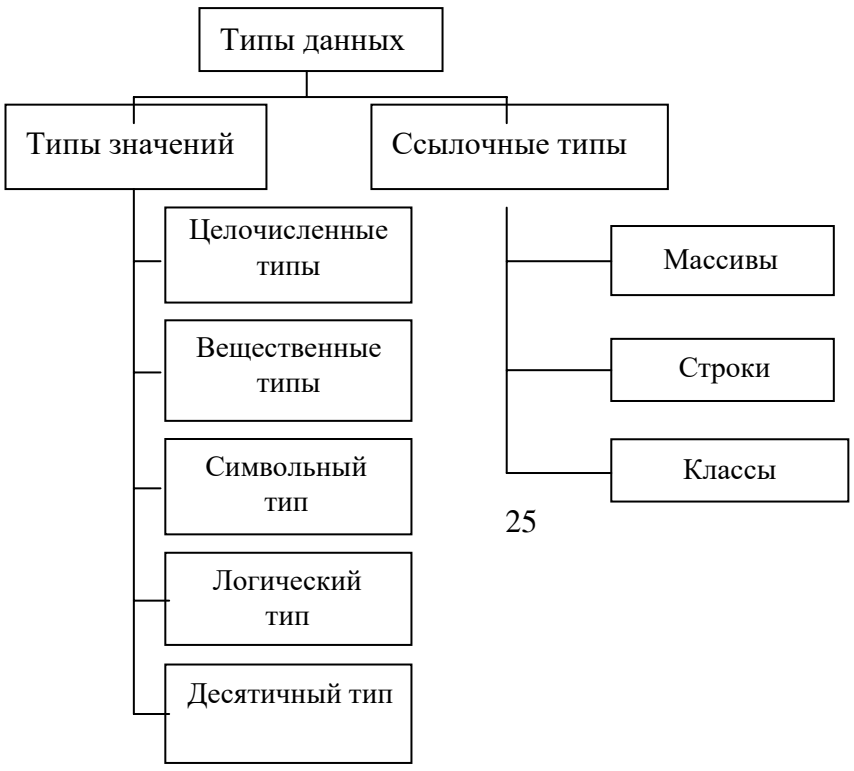


Рисунок 1 – Структура типов данных

Целые типы

В *C#* определены восемь целочисленных типов: *byte, sbyte, short, ushort, int, uint, long* и *ulong* (таблица 1).

Таблица 1 − Целочисленные типы

Тип	Описание	Диапазон	Примеры объявлений
sbyte	8-битовый знако-вый целый (1 байт)	-128 : 127	sbyte val = 12;

Тип	Описание	Диапазон	Примеры объявлений
short	16-битовый знако-вый целый (2 байта)	-32768 : 32767	short val = 12;
int	32-битовый знако-вый целый (4 байта)	-2 147 483 648 : 2 147 483 647	int val = 12;
long	64-битовый знако-вый целый (8 бай-тов)	-9223372036854775808: 9223372036854775807	long val1 = 12; long val2 = 34L;
byte	8-битовый беззна-ковый целый (1 байт)	0 : 255	byte val1 = 12;
ushort	16-битовый беззна-ковый целый (2байта)	0 : 65535	ushort val1 = 12;
uint	32- битовый беззна-ковый целый (4 байта)	0 : 4 294 967 295	uint val1 = 12; uint val2 = 34U;
ulong	64- битовый беззна-ковый целый (8 байтов)	0 : 18446744073709551615	ulong val1 = 12; ulong val2 = 34U; ulong val3 = 56L; ulong val4 = 78UL;

Вещественные типы

Вещественные типы (типы с плавающей точкой) позволяют представлять числа с дробной частью. В *C#* имеются две разновидности типов данных с плавающей точкой: *float* и *double*. Они представляют числовые значения с одинарной и двойной точностью соответственно (таблица 2).

Таблица 2 − Вещественные типы

Тип	Описание	Диапазон	Примеры объявлений

float	Вещественный с плавающей точкой с одинарной точностью (4 байта)	5E-45 до 3,4E+38	float val = 1.23F;
double	Вещественный с плавающей точкой с двойной точностью (8 байтов)	5E-324 до 1,7E+308	double val1 = 1.23; double val2 = 4.56D;

Десятичный тип данных

Для представления чисел с плавающей точкой высокойточности предусмотрен также десятичный тип decimal, которыйпредназначен для применения в финансовых вычислениях. Этот тип имеет разрядность 128 бит для представления числовых значений в пределах от 1E-28 до 7,9E+28. Для обычных арифметических вычислений с плавающей точкой характерны ошибки округления десятичных значений. Эти ошибки исключаются при использовании типа decimal, который позволяет представить числа с точностью до 28 (а иногда и 29) десятичных разрядов. Благодаря этому тип данных decimal способен представлять десятичные значения без ошибок округления, он особенно удобен для расчетов, связанных с финансами

Символы

В *C#* символы представлены не 8-разрядным кодом, как во многих других языках программирования, например *C++*, а 16- разрядным кодом, который называется юникодом (*Unicode*). В юникоде набор символов представлен настолько широко, что он охватывает символы практически из всех естественных языков на свете. В *C#* определен тип *char*, представляющий 16-разрядные значения без знака в пределах от 0 до 65 535. При этом стандартный набор символов в 8-разрядном коде *ASCII* является подмножеством юникода в пределах от 0 до 127. Следовательно, символы в коде *ASCII* по-прежнему остаются действительными в *C#*.

Логический тип данных

Тип *bool* представляет два логических значения: "истина" и "ложь". Эти логические значения обозначаются в *C#* зарезервированными словами *true* и *false* соответственно. Следовательно, переменная или выражение типа *bool* будет принимать одно из этих логических значений.

Строки

Основным типом при работе со строками является тип *string,* задающий строки переменной длины. Тип *string* представляет последовательность из нуля или более символов в кодировке
Юникод. Класс *String* в языке *C#* относится к ссылочным типам. Над строками - объектами этого класса - определен широкий набороопераций, соответствующий современному представлению о том, как должен быть устроен строковый тип. По сути, текст хранится в виде последовательной доступной только для чтения коллекцииобъектов *Char*.

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

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

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

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

Несмотря на то что тип char целочисленный, его не следует путать со всеми остальными целочисленными типами.

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

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

В *C#* литералами называются постоянные значения, представ-ленные в удобной для восприятия форме. Например, число 100 яв- ляется литералом. У литералов должен быть также конкретныйтип, поскольку *C#* является строго типизированным языком. По умолчанию литералы с фиксированной точкой относятся к типу int.

Для указания типа long к литералу присоединяется суффикс *l* или *L.* Например, 12 — это литерал типа *int*, а *12L* — литерал типа *long*.Для указания целочисленного типа без знака к литералу присоеди- няется суффикс *u* или *U*. Следовательно, 100 — это литерал типа *int*, а 100U — литерал типа *uint*. А для указания длинного целочис- ленного типа без знака к литералу присоединяется суффикс *ul* или *UL.* Например, 984375UL — это литерал типа *ulong*.

Кроме того, для указания типа *float* к литералу присоединяет- ся суффикс *F* или *f.* Например, 10.19F — это литерал типа *float*. Можете даже указать тип *double*, присоединив к литералу суффикс *d* или *D*, хотя это излишне. Ведь, как упоминалось выше, по умол- чанию литералы с плавающей точкой относятся к типу *double*.

И, наконец, для указания типа *decimal* к литералу присоеди- няется суффикс *m* или *M.* Например, 9.95M — это десятичный ли- терал типа *decimal*.

Математические функции

Для работы с данными могут использоваться математическиефункции, представленные ниже (таблица 3). Для тригонометриче- ских функций угол задается в радианах.

Таблица 3 − Стандартные математические функции

Функция	Описание	Примеры
Math.Abs	Возвращаем абсолютное число, имеет 7 перегрузок, то есть метод принимает разные типы перемен- ных.	int i = Math.Abs(x);
Math.Acos	Арккосинус. Определяется угол, косинус которого равен указан- ному числу.	double i =Math.Acos(0.5);
Math.Asin	Арксинус. Определяется угол, синус которого равен указанному числу.	double i = Math.Asin(0.5);

Math.Atan	Арктангенс. Возвращает угол, тангенс которого равен указан- ному числу.	double i = Math.Atan(0.5);
Math.Cos	Возвращает косинус угла.	double x =Math.Cos(1.4);
Math.Exp	Экспоненциальная функция e^x	double x = Math.Exp(2);

Функция	Описание	Примеры
Math.Log	Вычисление логарифма. X - аргумент, Osn - основание лога- рифма.	double y = Math.Log(X,Osn);
Math.Log10	Вычисление десятичногологарифма.	Double x = Math.Log10(10)
Math.Max	Возвращает из 2-х чисел большее число. Имеет 11 перегруженыхметодов.	Int x = Math.Max(10,20);
Math.Min	Возвращает из 2-х чисел мень- шее число. Имеет 11 перегруженых методов.	int x = Math.Min(10,20);
Math.PI	Возвращает число Пи	Double pi = Math.PI;
Math.Pow	Вычисляет число, возведенное в степень: a^x	double i = Math.Pow(a, x);
Math.Sin	Возвращает синус угла.	double p = Math.Sin(0.5);
Math.Sqrt	Возвращает квадратный корень.	double r = Math.Sqrt(7);
Math.Tan	Возвращает тангенс угла.	Double p = Math.Tan(1.04);
Math.Floor	Возвращает наибольшее целое число, которое меньше или равно заданному числу с плавающей запятой двойной точности.	double p = Math. Floor (7.04); Ответ : 7
Math.Ceiling	Возвращает наименьшее целое число, которое больше или равно указанному числу.	double p = Math.Ceiling (7.04); Ответ : 8

Оператор присваивания

Оператор присваивания обозначается одиночным знаком ра- венства (=). В *C#* оператор присваивания действует таким же обра- зом, как и в других языках программирования. Ниже приведена егообщая форма:

имя_переменной = выражение;

Здесь имя_переменной должно быть совместимо с типом вы- ражения. У оператора присваивания имеется одна интересная осо- бенность, о которой полезно знать: он

позволяет создавать цепочкуопераций присваивания. Рассмотрим следующий фрагмент кода:

```
int x, y, z;
x = y = z = 10; // присвоить значение 10 переменным x, y и z
```

В приведенном выше фрагменте кода одно и то же значение 10 задается для переменных x, y и z с помощью единственного оператора присваивания. Это значение присваивается сначала пе- ременной z, затем переменной y и, наконец, переменной x. Такой способ присваивания "по цепочке" удобен для задания общего зна- чения целой группе переменных.

Укороченные операторы присваивания

В *C#* предусмотрены специальные *укороченные* (составные) операторы (таблица 4) присваивания, упрощающие программиро- вание некоторых операций присваивания. Обратимся сначала к простому примеру:

```
x = x + 3;
// Можно переписать следующим образомx += 3;
```

Пара операторов += указывает компилятору на то, что пере- менной x должно быть присвоено ее первоначальное значение, увеличенное на 3.

Таблица 4 − Укороченные операторы присваивания

Оператор	Аналог
x += 1	x = x + 1;
x -= 1	x = x - 1;
x *= 1	x = x * 1;
x /= 1	x = x / 1;
x %= 1	x = x % 1;
x |= 1	x = x | 1;
x ^= 1	x = x ^ 1;

У укороченных операторов присваивания имеются два глав- ных преимущества. Во-первых, они более компактны, чем их "не- сокращенные" эквиваленты. И, во-вторых, они дают более эффек- тивный исполняемый код, поскольку левый операнд этих операторов вычисляется только один раз. Именно по этим причинам со- ставные операторы присваивания чаще всего применяются в про- граммах, профессионально написанных на *C#*.

Операции присваивания возвращают как результат присвоен- ное значение, поэтому допускается сцепление

Операции инкремента + + и декремента - -

Операции инкремента (+ +) и декремента (- -) сводятся к уве- личению (+ +) или уменьшению (- -) операнда на единицу. Эти операции выполняются быстрее, чем обычное сложение и вычита- ние.

Если операция инкремента или декремента помещена перед переменной, говорят о префиксной форме записи (++n или - -n). Если операция инкремента или декремента записана после пере- менной, то говорят о постфиксной форме записи (n++ или n- -). При префиксной форме переменная сначала увеличивается или уменьшается на единицу, а затем ее новое значение используется. При постфиксной форме в выражении используется текущее зна- чение переменной, а после ее значение изменяется на единицу. Пример:

1) int i=1;
int j = i+ + *5;

Результат:

j = 1*5 = 5
i = 2

2) int i=1;
int j = + +i * 5;

Результат:

j = 2*5 = 10
i = 2

Рассмотрим приоритеты операций (таблица 5).Таблица 5 − Приоритеты операций

Приоритет	Категория	Операции	Порядок
0	Первичные	(expr); x.y; f(x); a[x]; x++; x new; sizeof(t); typeof(t); checked(expr); unchecked(expr)	Слева направо
1	Унарные	+ - ! ~ ++x --x (T)x	Слева направо
2	Мультипликативные (Умножение)	- * / %	Слева направо
3	Аддитивные (Сложение)	+ -	Слева направо
4	Сдвиг	<< >>	Слева направо
5	Отношения, проверка типов	< > <= >= is as	Слева направо

Приоритет	Категория	Операции	Порядок
6	Эквивалентность	= = !=	Слева направо
7	Логическое И	&	Слева направо
8	Логическое исключающее ИЛИ (XOR)	^	Слева направо
9	Логическое ИЛИ (OR)	|	Слева направо
10	Условное И	&&	Слева направо
11	Условное ИЛИ	||	Слева направо
12	Условное выражение	? :	Справа налево
13	Присваивание	= *= /= %= += -= <<= >>= &= ^= |=	Слева направо

Пример программирования линейного алгоритма
Задание. Даны действительные числа x, y, z.

Составить программу вычисления для заданных значений x, y, zарифметического выражения

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

1. На рисунке 2 – разработка алгоритма:

– входные данные: x, y, z – действительные числа;

– выходные данные: u– действительное число.

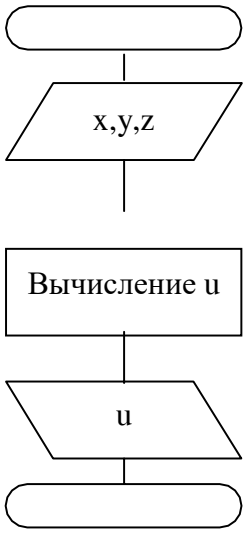
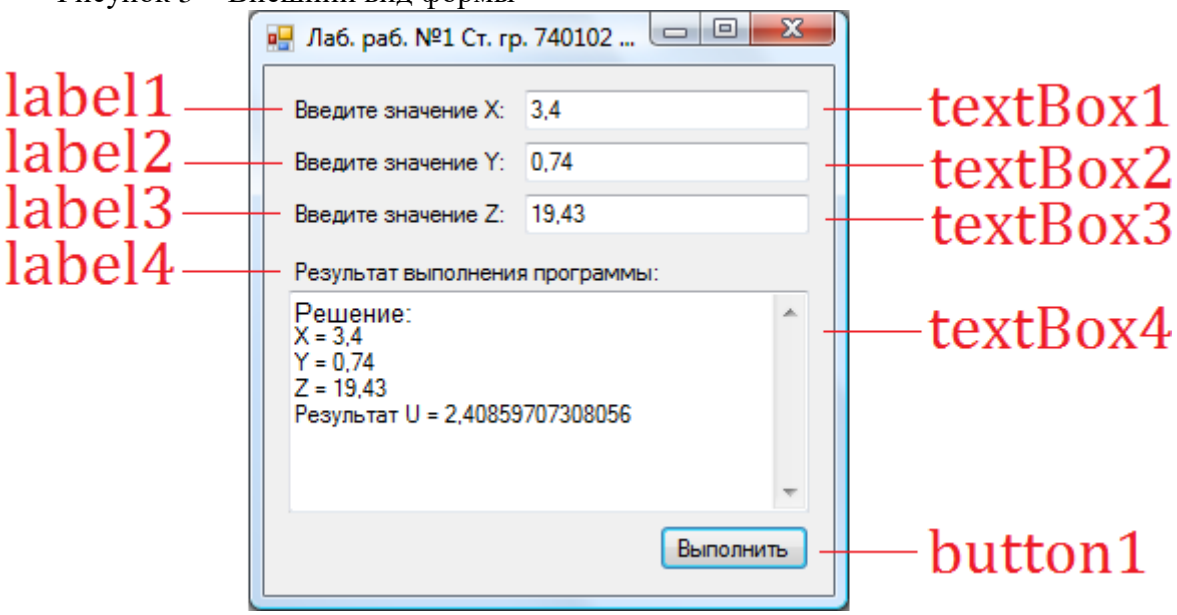


Рисунок 2 − Схема алгоритма решения задачи

2. На рисунке 3 – разработка формы.

Рисунок 3 − Внешний вид формы



Для вывода результатов работы программы используется тек- стовое окно, которое представлено элементом управления *textBox*. 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* вокне появится вертикальная, а при необходимости и горизонталь- ная полосы прокрутки.

Информация, которая отображается построчно в окне, нахо- дится в массиве строк *Lines*, каждая строка которого имеет тип *string*. Однако напрямую нельзя обратиться к этому свойству для добавления новых строк, поскольку размер массивов в *C#* опреде- ляется в момент их инициализации. Для добавления нового эле- мента используется свойство *Text,* к текущему содержимому кото- рого 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 отли- чаться в разных операционных системах и потому представлен свойством класса *Environment*) и сама новая строка. Если добавля- ется числовое значение, то его предварительно нужно привести в символьный вид методом *ToString()*.

Работа с программой происходит следующим образом. На- жмите (щелкните мышью) кнопку “Выполнить”. В окне *textBox4* появляется результат. Измените исходные значения x, y, z в окнах *textBox1 – textBox3* и снова нажмите кнопку ”Выполнить” - появит- ся новые результаты.

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)
{
// Вывод строки в многострочный редакторtextBox4.Text = "Решение:";
}
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();
}
}
}
```

Индивидуальные задания
Решите две задачи из первого и второго уровней сложности.
Задачи первого уровня сложности
Вычислите значения f и g, если:

Вариант	f	g
1	$3\sin(2^x)+0.35x^3-3.8$	$\ln(x)-x+1.8$
2	$0.25\ x^4+3x+2.5$	$x^8 * \operatorname{tg} x-1/3$
3	$\sin(\ln\|x\|)-\cos(\ln\|x\|)$	$\operatorname{tg}(x/2)-\operatorname{ctg}(x/2)+x$
4	$x-2+\sin(1/x)$	$\sqrt[3]{x}$ $0.4x^{12}+\operatorname{arctg}(\quad)-x$
5	$e^{2x+4}+\ln\|x\|-10\ x^5+3x$	$1-2x^7-\cos(3x)$
6	$\cos(2x)-5x^9+1$	$3.6^x-2.3x^5-3$
7	$1-x^7+\sin x-\ln\|1+x\|$	$3x-\quad 3.7\sqrt{x}+e^{x+5}$
8	$3\ln(x^2)+6\ln\|x\|-6$	$x+3.6x-\sin x^7$
9	$x-2x*\sin x-\cos x^9$	$\sqrt[3]{x}$ $\cos(1.3x)+\quad *\ln x$
10	$x^2-\ln\|1+x\|-3$	$\ln(x^2+5)-\cos^3 x$
11	$x-1/(3+\sin 3.6x)$	$1/(x^6+13x)-\operatorname{arctg} x$
12	$0.1x^4-x*\ln\|x\|$	$\cos(\operatorname{arctg}^5 x)-3$
13	$\operatorname{tg} x-3\operatorname{tg}^2 x+0.2\operatorname{tg}^3 x$	$3*\ln(x^4+2.8)-x$
14	$\arccos x-\quad \|1 \square 0.3x\|$	$1/\operatorname{arctg} x+\cos x-4.2$
15	$3x^5-4\ln\|x-5\|$	$(1-2x)/(x^3+5.4)$

Вариант	f	g
16	$\cos(2/x)-2\sin(1/x)$	$\operatorname{arctg} x-\sin^2 x+5$
17	$e^x-e^{-x+2}+5\ln\|x-3\|$	$13x^6+13\sin x+13\cos x$
18	$\sqrt[3]{x}$ $\sin(\ln\|x\|)+$	$\ln 0.7x-\cos^7 x$

19	$1 + \sin^2(x + y)$	$2 + \lvert x - 2x/(1 + x^y)\rvert$
20	$\cos^2(\mathrm{arctg}(1/y))$	$2\,\lvert\cos(x - /6)\rvert$
21	$1/(2x) + \sin(y)$	^{5} x □ lnx
22	$y^2 + \lvert x^2/y + x/4\rvert$	^{3} $5x$ □ lg x □ 8.2
23	$5\lg(x^3) + 3.2\lg\lvert x\rvert - 2$	$7.2^x - 5x - 9.8$
24	^{3} x □ lg | x |	$\lvert y^5 - x + \sin x^2\rvert$
25	^{3} 2x □sin | x |	$5*\lg(x^3+9.8) - x^7$
26	^{7} e^{3x-5} □ | $\mathrm{tg}x^3$ |	$2\,\lvert\sin(x^3 - /3)\rvert$
27	$2x^5 + \lvert x + 7x/(\mathrm{tg}^3 x + x^y)\rvert$	^{5} $4x^3$ □ | $\cos^9 x^7$ |
28	^{9} $3x^2$ □ |3tgx |	$\lg(3x^2+6x+81) - y^8$
29	$\ln(xy^2+6^x+31x)+71.5^x$	^{3} $4x^5$ □ lg |x |
30	$5\ln(x^3) + 7.2\log_2\lvert x\rvert/3$	^{3} $5e^{9x-5}$ □ | $\lg^7 x^3$ |

ПРАКТИЧЕСКАЯ РАБОТА № 5. Кортежи

Массивы

1.2.1. Создание и вывод массивов

Создадим проект консольного приложения C#, название SharpArrays.

Метод Main.

Сначала создадим прямоугольный массив:

```
// размер массива
const int size = 5;
int[,] matrix = new int[size, size];
// заполняем массив
for (int i = 0; i < size; i++) {
for (int j = 0; j < size; j++) { matrix[i, j] = j + (i * size);
}
}
// выводим массив
for (int i = 0; i < size; i++) {
for (int j = 0; j < size; j++) { Console.Write(matrix[i, j] + "\t");
}
Console.WriteLine();
}
```

Затем создадим ломаный массив:

```
// ломаный массив
int[][] M = new int[size][];
// создаем массивы
```

```
for (int i = 0; i < M.Length; i++) {M[i] = new int[i + 1];
    }
    // выводим массивы
    for (int i = 0; i < size; i++) {
    Console.Write("Строка {0} : {1}:\t", i, M[i].Length);for (int j = 0; j < M[i].Length; j++) {
    Console.Write(M[i][j] + " ");
    }
    Console.WriteLine();
    }
```

Кортежи.

Научиться использовать приемы обработки кортежей при решении практических задач.

Ход выполнения работы:

1. №1. Дополните приведенный код, так чтобы он вывел элементы кортежа countries, кроме первых двух. Результатом вывода должна быть строка ('Slovakia', 'Canada', 'Slovenia', 'Italy', 'Spain', 'Ukraine', 'Chile', 'Cameroon'). Исходный программный код: countries = ('Russia', 'Argentina', 'Slovakia', 'Canada', 'Slovenia', 'Italy', 'Spain', 'Ukraine', 'Chile', 'Cameroon')
2. №2. Дополните приведенный код так, чтобы переменная new_tuples, содержала список кортежей на основе списка tuples с последним элементом каждого кортежа, замененным на численное значение 100. Исходный программный код: tuples = [(10, 20, 40), (40, 50, 60), (70, 80, 90), (10, 90), (1, 2, 3, 4), (5, 6, 10, 2, 1, 77)] …
3. №3. Дополните приведенный код так, чтобы он вывел произведение элементов кортежа numbers. Исходный программный код: numbers = (-6, 41, 43, 47, 53, 59, 61, -96, 71, 1000, -1) … # должно получиться: -646381203181632000

ПРАКТИЧЕСКАЯ РАБОТА № 6. Создание классов в C#

Цели:

- классы C#.

Задачи:

- создание структур и классов;
- наследование и полиморфизм;
- пространства имен.Опорные документы:

2.1. Определение структуры

Создадим проект консольного приложения C#.

Название проекта — SharpStruct.

Структуры в C# являются структурным типом данных, но при этом могут обладать методами. Объявим перед классом программы перечисле- ние типов работников:

```
enum emp_type : byte {
manager = 10, grunt = 1, contractor = 100, VP = 9
}
```

Заметим, что мы указываем тип данных перечисления byte.

Ниже объявим структуру, описывающую работника:

```
struct sEmployee {
public emp_type title;public string name; public short deptID;
}
```

Структура C# может иметь конструктор, но только с параметрами.Описываем его в структуре sEmployee:

```
// конструктор
public sEmployee(emp_type et, string n, byte d) {title = et;
```

name = n; deptID = d;
}
Опишем метод для вывода информации о работнике (перед Main):
void DisplaysEmloyeeStats(sEmployee e) { Console.WriteLine("{0}: отдел {1}, тип {2}",
e.name, e.deptID,
Enum.Format(typeof(emp_type), e.title, "G") + ".");
}
В Main создаем представителя класса программы и работника bill:
Program t = new Program();
sEmployee bill = new sEmployee(emp_type.VP, "Bill", 10);t.DisplaysEmloyeeStats(bill);
Запустим программу, убедимся, что сведения выводятся.

Для преобразования структурного типа в ссылочный тип используетсяупаковка. Упаковка выполняется просто: присвоением структурного типа ссылочному типу (метод Main):

// упаковка структурного типа в ссылочный
object bill_inbox = bill;

Опишем еще один метод класса программы, который распаковывает ссылочный тип в структурный тип. Распаковка также выполняется просто

— приведением к структурному типу:

void UnboxsEmployee(object o) {
// распаковываем ссылочный тип в структурный sEmployee e = (sEmployee)o;
DisplaysEmloyeeStats(e);
}
В функции Main вызываем новый метод дважды:
// упакованный
t.UnboxsEmployee(bill_inbox);
// компилятор производит упаковку автоматически
t.UnboxsEmployee(bill);

2.2. Определение класса

Создадим проект консольного приложения C#, название SharpClass.
Класс в C# определяется примерно так же, как и в C++:
class Employee {
// закрытые данные классаprivate string fullName; private short empID; private double currPay;
}

Если в классе определен конструктор с параметрами, определениеконструктора по умолчанию (без параметров) является обязательным:

// конструктор по умолчанию
public Employee() {}
// конструктор преобразования
public Employee(string fullName, short empID, double currPay) {this.fullName = fullName;
this.empID = empID; this.currPay = currPay;
}
Методы класса определяются обычным образом:
// метод для увеличения зарплаты
public void GiveBonus(double amount) {currPay += amount;
}

А это виртуальный метод:

```
// выводит сведения о текущем состоянии
public virtual void DisplayStats() { Console.WriteLine("\nИмя: {0}", fullName);
Console.WriteLine("Зарплата: {0}", currPay); Console.WriteLine("Табельный номер: {0}", empID);
}
```

Код метода Main определяет двух сотрудников:

```
static void Main(string[] args) {
Employee e = new Employee("Bill", 100, 30000.0);e.GiveBonus(500.0);
e.DisplayStats();
}
```

Запускаем программу, убеждаемся, что сведения выводятся.Второй сотрудник:

```
static void Main(string[] args) {
Employee e = new Employee("Bill", 100, 30000.0);e.GiveBonus(500.0);
e.DisplayStats();Employee e2;
e2 = new Employee("Mary", 101, 36000.0);e2.GiveBonus(1000.0);
e2.DisplayStats();
}
```

2.3. Внутренние классы

Нововведение C#, связанное с организацией .NET — внутренние классы и элементы. Внутренние классы можно использовать только внутритекущей сборки.

Перед классом Employee опишем один такой внутренний класс:

```
internal class EmployeeHelper {private string helpstring; public EmployeeHelper() { }
public EmployeeHelper(string hs) {helpstring = hs;
}
}
```

Внутренними могут быть также структуры, перечисления, методы и свойства.

Опишем в элементах данных класса Employee ссылку на класс EmployeeHelper:

```
class Employee {
// закрытые данные класса private EmployeeHelper helper;private string fullName;
. . .
}
```

В этом нет никакого смысла, просто так…

2.4. Свойства в классах

Новым является также возможность определять не только методы классов, но свойства. Свойство задается процедурами get (чтение) и set (запись нового значения). Их еще называют аксессорами (от access) или жегеттерами и сеттерами. Новое значение свойства при этом указываетсяспециальным словом value.

Опишем свойство Name для класса Employee:

```
// свойство Имя
public string Name {get {
return fullName;
}
set {
fullName = value;
}
}
```

Используем свойство Name в функции Main:

```
static void Main(string[] args) {
Employee e = new Employee("Bill", 100, 30000.0);e.GiveBonus(500.0);
e.DisplayStats();Employee e2;
e2 = new Employee("Mary", 101, 36000.0);e2.GiveBonus(1000.0);
e2.DisplayStats();
// используем свойство классаe2.Name = "Ann"; e2.DisplayStats();
. . .
}
```

Обратим внимание на отсутствие скобок () после имени свойства.Самостоятельно опишем свойство Salary (зарплата).

Свойства могут быть только для чтения и только для записи.Пример свойства только для чтения:

```
// свойство табельный номер только для чтения
public short ID {get {
return empID;
}
}
```

(Заметим, что в современной версии C# свойства, описанные простым способом (не имеющие алгоритмов), можно указывать только именами ак- сессоров, а код можно не писать.)

В функции Main выводим значение свойства для e2:

```
Console.WriteLine("Табельный номер e2: {0}", e2.ID);
```

Запустим программу.

2.5. Статический конструктор

Еще одно нововведение C# — статические конструкторы. Их единст-венное назначение — задавать значения статическим элементам данных.

Определим в классе Employee статический элемент данных, задающийназвание компании:

```
class Employee {
// название организации
private static string company_name;
// название организации
private static string company_name;
. . .
}
```

Опишем статический конструктор в классе Employee:

```
// статический конструктор
static Employee() { company_name = "ОТИ МИФИ";
}
```

Опишем метод, возвращающий название организации:

```
// метод возвращает название организации
public string GetCompanyName() {return company_name;
}
```

В методе Main выведем в консоль название организации:

```
Console.WriteLine("Метод: {0}", e.GetCompanyName());
```

Мы можем также определить статическое свойство в классе Employee:

```
// мвойство возвращает название организации
```

public static string Company {get {

return company_name;

}

}

В конце метода Main используем это свойство:

Console.WriteLine("Свойство: {0}", Employee.Company);

ПРАКТИЧЕСКАЯ РАБОТА №7. ОПЕРАЦИИ: АРИФМЕТИЧЕСКИЕ, ЛОГИЧЕСКИЕ. ОПЕРАЦИИ НАД СТРОКОВЫМИ И СИМВОЛЬНЫМИ ДАННЫМИ

Арифметические операции. Задача: напишите программу на C#, которая рассчитывает абсолютную разность между n и числом 48, где n — это любое целое число. Если n больше 48, то верните тройную абсолютную разность.

Логические операции.

Вычислить значение логического выражения при следующих значениях логических величин X, Y и Z: X = Истина, Y = Истина, Z = Ложь:
а) не X и Y; б) X или не Y; в) X или Y и Z.

Вычислить значение логического выражения при следующих значениях логических величин A, B и C: A = Истина, B = Ложь, C = Ложь:
а) A или B и не C; г) A и не B или C;
б) не A и не B; д) A и (не B или C);
в) не (A и C) или B; е) A или (не (B и C)).

1.3. Строки

1.3.1. Форматирование строк

Создадим проект консольного приложения C#, название SharpStrings.

Для вывода данных используются форматы преобразований, обозна-чаемые буквами. Метод Main.

Выведем в консоль несколько чисел в разных форматах:

static void strings() {

Console.WriteLine(“C format: {0:C}”, 99989.987); Console.WriteLine(“D9 format: {0:D9}”, 99999); Console.WriteLine(“E format: {0:E}”, 99999.76543); Console.WriteLine(“F format: {0:F3}”, 99999.9999); Console.WriteLine(“N format: {0:N}”, 99999); Console.WriteLine(“X format: {0:X}”, 99999); Console.WriteLine(“x format: {0:x}”, 99999);

}

Например, буква C означает Currency (валюта), D9 — девять цифр, E,
как и следовало ожидать — экспоненциальный формат, и т.п.

Форматирование выполняет также метод Format класса String с полу-чением новой строки:

string s = String.Format(“На счету {0:C}”, 99989.987);Console.WriteLine(s);

В качестве параметров строки вывода могут использоваться массивы,тогда индексы в блоках {} соответствуют индексам элемента массива.

Сначала нужно создать массив типа object:

object[] stuff = { «Hi», 2.9, 1, «there», «83» };
Console.WriteLine(«stuff: {0}, {1}, {2}, {3}, {4}», stuff);

1.3.2. Методы класса System.String

Строки в C# являются встроенным типом данных, и они имеют коди- ровку Unicode. Строки происходят от System.String, и операции со строка- ми — это методы и свойства: Length — возвращает длину строки, Concat

— объединяет две строки в одну, CompareTo — сравнивает одну строку с другой, Copy — возвращает копию строки, и другие …

Строки — ссылочный тип данных, однако при их сравнении при по- мощи операций «==» и «!=» сравниваются значения строк, а не адреса об- ластей памяти. Операция + перегружена так, что она автоматически вызы- вает статический метод Concat (конкатенация).

Описываем две строки и сравниваем их:

System.String strObj = “test;”;string s1 = “TEST”;

// сравниваем значения строк

if (s1 == strObj) { Console.WriteLine(«Одно и то же»);

} else {

Console.WriteLine(«Разные строки»);

}

Убеждаемся, что строки разные. Конкатенация выполняется просто:

// конкатенация

string newstr = strObj + s; Console.WriteLine(“strObj + s = {0}”, newstr);

Для доступа к элементам строки используется индексатор, как в C:

// индексатор для доступа к отдельному символу строки

for (int I = 0; I < s.Length; i++) { Console.WriteLine(“Char {0} is {1}”, I, s[i]);

}

1.3.3. Управляющие последовательности

Управляющие последовательности практически такие же, как в C:

// управляющие последовательности

string a, b, c;

a = “Применение \”кавычек\””;Console.WriteLine(a);

b = “Применение \”\\\\\” и \”\\n\” – C:\\app\n.file”;Console.WriteLine(b);

// @ отменяет действие управляющих последовательностей c = @»Последовательности отменены – C:\app\n.file»; Console.WriteLine©;

В C#, как и в Java, значение строки нельзя изменить. Методы класса System.String возвращают копии строк и с этим ничего нельзя сделать.

// строки не изменяются

System.String fixeds = “string-String”;Console.WriteLine(fixeds);

// метод ToUpper возвращает копию строки string uppers = fixeds.ToUpper(); Console.WriteLine(uppers); Console.WriteLine(fixeds);

1.3.4. Построитель строк

Для работы со строками без создания копий нужен StringBuilder:

// StringBuilder работает с одной строкой

StringBuilder buff = new StringBuilder(«буфер строки»); buff.Append(«, который стал длиннее»); Console.WriteLine(buff);

После получения нужной строки используем метод ToString чтобыперевести содержимое объекта StringBuilder в обычный тип данных string:

// получим string из StringBuilder string e = buff.ToString().ToUpper(); Console.WriteLineI; Console.WriteLine(buff);

1.4. Перечисления

Создадим проект консольного приложения C#.

Название проекта — SharpEnums.

Базовый тип System.Enum имеет методы и свойства, которые могутоблегчить работу с перечислениями. Определим следующее перечисление:

```
namespace SharpHello {enum emp {
    manager = 1,
    grunt = 10,
    contractor = 100,
    VP = 99
    }
    class Program {
```

В Main выводим тип данных перечисления:

```
// тип данных перечисления
Console.WriteLine(Enum.GetUnderlyingType(typeof(emp)));
```

При помощи метода Enum.Format можно получить информацию о пе-ременной перечисления:

```
// информация о переменной перечисления
Console.WriteLine(«bill – {0}», Enum.Format(typeof(emp), bill, «G»));
Console.WriteLine(“bill – {0}”, Enum.Format(typeof(emp), bill, “X”));
Console.WriteLine(“bill – {0}”, Enum.Format(typeof(emp), bill, “d”));
```

Флаг форматирования G означает «вывести как строку».

При помощи метода GetValues можно получить массив элементов пе-речисления:

```
// информация об элементах перечисления
Array a = Enum.GetValues(typeof(emp)); Console.WriteLine(«элементов {0}.», a.Length);
int i = 0;
foreach (emp e in a) {
Console.WriteLine(“Название элемента {0} – {1}”, i++,Enum.Format(typeof(emp), e, “G”));
}
```

Свойство IsDefined:

```
// определяет наличие элемента перечисления
if (Enum.IsDefined(typeof(emp), “VP1”)) {Console.WriteLine(“Существует.”);
} else {
Console.WriteLine(«Нет такого слова в перечислении.»);
}
```

ПРАКТИЧЕСКАЯ РАБОТА №8. ССЫЛОЧНЫЕ ТИПЫ ДАННЫХ

1.5. Структурные и ссылочные типы

Создадим проект консольного приложения C#, название SharpSAC.

Типы C# делятся на структурные типы и ссылочные типы.

К структурным типам относятся числовые типы, перечисления и структуры. Память для них выделяется из стека метода. При копировании структурного типа создается побитовая копия.

К ссылочным типам относятся классы и интерфейсы (а также строки). Память для них выделяется из кучи. При копировании ссылочного типа создается еще одна ссылка на тот же объект.

Опишем следующие структуру и класс:

```
struct sfoo {
public int x;
}
```

```
class cfoo {
public int x = 100;
}
```

Первое отличие структуры от класса — в структуре нельзя инициали- зировать элементы данных, в классе можно (и нужно). Структуры и классыC# отличаются тем, что каждый элемент имеет свой модификатор доступа.

В методе Main проведем тесты со структурой sfoo:

```
sfoo s1; s1.x = 100;
// копия типа
sfoo s2 = s1;
// выводим структуры s1 и s2 Console.WriteLine("s1.x={0} s2.x={1}", s1.x, s2.x);
// изменим s2s2.x = 555;
// снова выводим структуры s1 и s2 Console.WriteLine("s1.x={0} s2.x={1}", s1.x, s2.x);
```

Запустим программу. Убедимся, что объект s2 изменился.

Такие же тесты нужно выполнить с классом cfoo, называя переменные c1 и c2:

```
// классы
cfoo c1 = new cfoo();
// копия типа
cfoo c2 = c1;
// выводим структуры c1 и c2 Console.WriteLine("c1.x={0} c2.x={1}", c1.x, c2.x);
// изменим c2c2.x = 555;
// снова выводим структуры c1 и c2 Console.WriteLine("c1.x={0} c2.x={1}", c1.x, c2.x);
```

Второе отличие структуры от класса — структуру не обязательно соз- давать при помощи new, а представитель класса создается только при по- мощи new.

1.6. Все есть объект

Создадим проект консольного приложения C#.

Название проекта — SharpObjects.

В .NET все есть объект, производный от System.Object, и все объектыимеют методы ToString, GetHashCode, GetType, Equals.

В Main выводим свойства разных объектов:

```
static void isobject() {int i = 128;
Console.WriteLine("{0}", i.ToString()); Console.WriteLine("{0}", 256.ToString());
Console.WriteLine("{0}", 256.GetTypeCode()); Console.WriteLine("{0}", i.GetType());
Console.WriteLine("{0}", i.GetTypeCode()); Console.WriteLine("{0}", int.MaxValue);
Console.WriteLine("{0}", int.MinValue); Program p1 = new Program();
// Информация об объекте
Console.WriteLine("ToString: {0}", p1.ToString());
Console.WriteLine("HashCode: {0}", p1.GetHashCode());
Console.WriteLine("Type: {0}", p1.GetType());
// создаем еще одну ссылку на p1Program p2 = p1;
// оба экземпляра указывают на одну область памяти?if (p1.Equals(p2)) {
Console.WriteLine("Тот же экземпляр!");
}
}
```

1.6.1. Замещение методов SystemObject

Создадим проект консольного приложения C#.

Название проекта — SharpObjectOverride.

Описываем класс Person:

```
namespace SharpObjectOverride {class Person {
    public string name = "";
    public string ssn = "";   // номер социального страхования
    public byte age = 1;public Person() {}
public Person(string n, string s, byte a) {name = n;
    ssn = s;age = a;
    }
    }
    struct sfoo {
```

Замещаем в классе Person методы Object:

```
public override string ToString() { StringBuilder sb = new StringBuilder();sb.Append("[Name =
    " + this.Name); sb.Append(" SSN = " + this.SSN); sb.Append(" Age = " + this.age + "]");
    return sb.ToString();
    }
public override bool Equals(object o) {Person temp = (Person)o;
    bool eq = temp.Name == this.Name;eq &= temp.SSN == this.SSN;
    eq &= temp.age == this.age;if (eq) {
    return true;
    } else {
    return false;
    }
    }
public override int GetHashCode() {return SSN.GetHashCode();
    }
```

Метод Main:

```
    Person p1 = new Person("Fred", "222-22-2222", 98);Person p2 = new Person("Fred", "222-
    22-2222", 98);
if (p1.Equals(p2) && p1.GetHashCode() == p2.GetHashCode()){ Console.WriteLine("P1 и P2
    одинаковы");
    } else {
    Console.WriteLine("P1 и P2 разные");
    }
    // меняем состояние p2p2.age = 2;
    // сравниваем заново
if (p1.Equals(p2) && p1.GetHashCode() == p2.GetHashCode()){ Console.WriteLine("P1 и P2
    одинаковы");
    } else {
    Console.WriteLine("P1 и P2 разные");
    }
    // используем замещенный метод ToString:Console.WriteLine(p1.ToString());
    // WriteLine вызывает метод ToString автоматически
    Console.WriteLine(p2);
```

Кроме виртуальных методов, System.Object имеет два статических ме-тода Equals и ReferenceEquals:

```
    // статические члены System.Object
    Person p3 = new Person("Sally", "333", 4);Person p4 = new Person("Sally", "333", 4);
```

```
// одинаково ли внутреннее состояние p3 и p4? Да! Console.WriteLine("P3 и P4 одинаковы: {0}",
object.Equals(p3, p4));
// представляют ли они один и тот же объект в памяти, Нет! Console.WriteLine("P3 и P4 одно и то же: {0}",
object.ReferenceEquals(p3, p4));
```

Запускаем программу, убеждаемся, что p3 и p4 одинаковы, но не одно и то же.

1.7. Статические методы класса Environment

Создадим проект консольного приложения C#.

Название проекта — SharpEnviron.

Метод Main:

```
// операционная система
Console.WriteLine("OS: {0}", Environment.OSVersion);
// текущий каталог
Console.WriteLine("Dir: {0}", Environment.CurrentDirectory);
// список логических дисков
string[] drives = Environment.GetLogicalDrives();for (int i = 0; i < drives.Length; i++) {
Console.WriteLine("Disk {0} : {1}", i, drives[i]);
}
// версия платформы .NET
Console.WriteLine(".NET: {0}", Environment.Version);
```

1.8. Контрольные вопросы и упражнения

1. Опишите сигнатуру метода Main.
2. Как получить доступ к аргументам командной строки?
3. Опишите вывод и форматированный вывод переменных.
4. Перечислите структурные и ссылочные типы.
5. Чем структурные типы отличаются от ссылочных?
6. Назовите общие методы всех объектов.

ПРАКТИЧЕСКАЯ РАБОТА № 9. МЕТОДЫ КЛАССА SYSTEM.ARRAY

1.9. Случайные числа

Создадим проект консольного приложения C#.

Название проекта — SharpRandom.

Описываем класс Teenager:

```
class Teenager {
// для генерации случайных чисел
private Random r = new Random();
}
```

Опишем метод Complain (в классе Teenager):

```
public string Complain() {
string[] messages = new string[5] {
"А я должен?", "Он первый начал...", "Я устал...","Ненавижу школу!", "Вы тааак заблуждаетесь!"
};
return messages[GetRandomNumber(5)];
}
```

Опишем метод GetRandomNumber (в классе Teenager):

```
public int GetRandomNumber(int upperLimit) {
// Random.Next() возвращает случайное целое число
// в диапазоне от 0 до upperLimitreturn r.Next(upperLimit);
}
```

Метод Main:

```
Teenager mike = new Teenager();
// mike, начинай жаловаться
for (int i = 0; i < 12; i++) { Console.WriteLine(mike.Complain());
}
```

Класс Array определяет много плезных методов: Clear — очищаетдиапазон массива, CopyTo — копирует массив в другой, Reverse — рас- ставляет элементы в обратном порядке, Sort — сортирует массив (если оп- ределен интерфейс IComparer), и другие … В Main создаем массив и пробуем найти элемент:

```
string[] N = new string[] { "123", "456", "189" };for (int i = 0; i < N.Length; i++) {
Console.WriteLine(N[i]);
}
// поиск
int index = Array.BinarySearch(N, "456"); Console.WriteLine("Найден элемент {0}", index);
```

Разворачиваем массив:

```
// развернем массив Array.Reverse(N); Console.WriteLine("Развернуто"); for (int i = 0; i < N.Length; i++) {
Console.WriteLine(N[i]);
}
```

Сортируем массив:

```
// сортировка
Array.Sort(N); Console.WriteLine("Сортировано"); for (int i = 0; i < N.Length; i++) {
Console.WriteLine(N[i]);
}
```

Очистим два элемента:

```
// очистим 2 элемента Array.Clear(N, 1, 2); Console.WriteLine("Очищено");
for (int i = 0; i < N.Length; i++) {Console.WriteLine(N[i]);
}
Console.WriteLine("Размер {0}", N.Length);
```

Практическая работа 10 Перегрузка операторов и методов

Цель занятия: Получить практические навыки создания перегруженных операторов и методов в классах

Перечень оборудования и программного обеспечения

Персональный компьютер Microsoft Office (Word, Visio) Microsoft Visual Studio

Краткие теоретические сведения

Полиморфизм - обеспечивает возможность задания различных реализаций некоторого единого по названию метода для классов различных уровней иерархий. Язык программирования поддерживает полиморфизм, если классы с одинаковой спецификацией могут иметь различную реализацию — например, реализация класса может быть изменена в процессе наследования

В C# допускается совместное использование одного и того же имени двумя или более методами одного и того же класса, при условии, что их параметры объявляются по-разному.

Существование в классе методов с одним и тем же именем называется перегрузкой, а сами одноименные методы называются перегруженными. Перегрузка методов полезна, когда требуется решать подобные задачи с разным набором аргументов. Перегрузка методов относится к одному из способов реализации полиморфизма в C#.

Перегрузка характерна и для знаков операций. В зависимости от типов аргументов, один и тот же знак может выполнять фактически разные операции. Классическим примером является знак операции сложения «+», который играет роль операции сложения не только для арифметических данных разных типов, но и выполняет конкатенацию строк.

Перегрузка требует уточнения семантики вызова метода. Когда встречается вызов неперегруженного метода, то имя метода в вызове однозначно определяет, тело какого метода должно выполняться в точке вызова. Когда же метод перегружен, то знания имени недостаточно - оно не уникально. Уникальной характеристикой перегруженных методов является их сигнатура. Перегруженные методы, имея одинаковое имя, должны отличаться либо числом аргументов, либо их типами, либо ключевыми словами (отметим: с точки зрения сигнатуры, ключевые слова **ref** и **out** не отличаются). Уникальность сигнатуры позволяет вызвать требуемый перегруженный метод.

Перегрузка широко используется при построении библиотеки FCL. Например, класс **Convert,** у которого 16 методов с разными именами, зависящими от целевого типа преобразования. Каждый из этих 16 методов перегружен, и в свою очередь, имеет 16 реализаций в зависимости от типа источника.

Пример.

Метод будет добавлять аргументы друг к другу и выводить результат:

```
public static void AddAndDisplay(char a, char b)
{
Console.WriteLine(a.ToString() + b.ToString());
}
public static void AddAndDisplay( bool a, bool b)
{
Console.WriteLine(a || b);
}

static void Main(string[] args)
{
AddAndDisplay(5, 8); // 13 AddAndDisplay('C', '#'); // "C#" AddAndDisplay(true, false); // "C#"
Console.ReadKey();
}
```

При вызове метода, компилятор сам решает, какой вариант метода AddAndDisplay необходимо вызвать.

Совершенно недостаточно, чтобы методы отличались только типами возвращаемых значений. Они могут также отличаться числом своих параметров. Когда вызывается перегружаемый метод, то выполняется тот его вариант, параметры которого соответствуют (по типу и числу) передаваемым аргументам.

Пример:

```
class GreetDemo
```

```
{
static void Greet( )
{
Console.WriteLine("Hello");
}
static void Greet(string Name)
{
Console.WriteLine("Hello " + Name)
}
static void Main( )
{
Greet( ); Greet("Alex");
}
}
```

Задания

1 Изучить теоретические сведения и задание к работе

2 Используя задание практической работы 22, создать конструкторы и методы класса

3 В соответствии с вариантом задания составить отлаженную программу.

Порядок выполнения работы

Изучить теоретические сведения и задание к работе

В соответствии с вариантом задания составить отлаженную программу.

Содержание отчета

1 Название работы
2 Цель работы
3 Технические средства обучения
4 Задания (условия задач)
5 Порядок выполнения работы
6 Листинг отлаженной программы в соответствии с вариантом задания
7 Ответы на контрольные вопросы
8 Вывод

Варианты заданий

Создать в классе методы, различающиеся типом возвращаемых значений и типами аргументов, содержащие перегрузку метода:

1. Сумма (одномерные массивы, логические значения, целые значения)
2. Разность (массивы, строки, целые значения)
3. Произведение (матрицы, логические значения, целые значения)
4. Пересечение (массивы, строки, прямоугольники)
5. Объединение (массивы, строки, прямоугольники)
6. Отрицание (логические значения, строки)
7. Сумма (вектора, логические значения, вещественные значения)

8. Разность (вектора, окружности, вещественные значения)
9. Произведение (вектора, логические значения, вещественные значения)
10. Пересечение (окружности, прямые)
11. Объединение (окружности, смежные углы)
12. Эквивалентность (массивы, логические значения, строки)
13. Сумма (вектора, логические значения, вещественные значения)
14. Произведение (вектора, логические значения, вещественные значения)
15. Пересечение (окружности, прямые)
16. Разность (массивы, строки, целые значения)
17. Произведение (массивы, логические значения, целые значения)
18. Разность (вектора, окружности, вещественные значения)
19. Отрицание (логические значения, строки)
20. Пересечение (окружности, прямые)
21. Сумма (одномерные массивы, логические значения, целые значения)
22. Разность (массивы, строки, целые значения)
23. Объединение (окружности, смежные углы)

Контрольные вопросы

1. Что такое полиморфизм?
2. Для чего предназначена перегрузка?
3. К каким элементам класса применяется перегрузка?
4. Какие характеристики оказывают влияние на идентификацию метода и могут быть перегружены?
5. Можно ли перегружать встроенные операции C#?
6. Общая форма перегрузки оператора.
7. Чем отличаются перегрузки унарного и бинарного операторов?

ПРАКТИЧЕСКАЯ РАБОТА №13-15.

Класс FileStream

Класс FileStream представляет возможности по считыванию из файла и записи в файл. Он позволяет работать как с текстовыми файлами, так и с бинарными. Он создаётся следующим образом:

FileStream fileStream = new FileStream(path, FileMode.Create);

Здесь в конструктор передается два параметра: путь к файлу и перечисление FileMode. Данное перечисление указывает на режим доступа к файлу и может принимать следующие значения:

Append: если файл существует, то текст добавляется в конец файл. Если файла нет, то он создается. Файл открывается только для записи.

Create: создается новый файл. Если такой файл уже существует, то он перезаписывается

CreateNew: создается новый файл. Если такой файл уже существует, то он приложение выбрасывает ошибку

Open: открывает файл. Если файл не существует, выбрасывается исключение

OpenOrCreate: если файл существует, он открывается, если нет - создается новый

Truncate: если файл существует, то он перезаписывается. Файл открывается только для записи.

Также можно использовать статические методы класса File, например
FileStream fileStream = File.Create(path); // создать новый файл
или
FileStream fileStream = File.Open(path); // открыть файл с доступом на чтение и запись
Для работы с бинарными файлами предназначена пара классов **BinaryWriter** и **BinaryReader**. Эти классы позволяют читать и записывать данные в двоичном формате.
Основные метода класса BinaryWriter
Close(): закрывает поток и освобождает ресурсы
Flush(): очищает буфер, дописывая из него оставшиеся данные в файл
Seek(): устанавливает позицию в потоке
Write(): записывает данные в поток
Объект BinaryWriter создаётся так:
BinaryWriter binaryWriter = new BinaryWriter(fileStream);
Для того, чтобы закрыть потоки и освободить ресурсы можно использовать метод Close(). Но удобнее использовать директиву using

```
using (FileStream fileStream = new FileStream(path, FileMode.Create))
using (BinaryWriter binaryWriter = new BinaryWriter(fileStream))
{
   // запись информации в файл
}
```

или можно сократить запись:

```
using (var binaryWriter = new BinaryWriter(File.Create(path)))
{
   // запись информации в файл
}
```

Бинарные файлы (запись)
Создайте метод
static void CreateTheNewFile(string path, params int[] values)
который записывает в файл целых чисел набор значений, перечисленных через запятую. **Указание** Есть разные виды ошибок, при котором работа с файлом невозможна, например отсутствие прав доступа к файлу. Чтобы программа не "вылетала", используйте конструкцию try...catch

```
try
{
   // код, вызывающий ошибку
}
catch (IOException e)
{
   Console.WriteLine($"Ошибка обработки файла {e.Message}");
}
```

Используйте try... catch в методе или поместите в блок try вызов метода.
Чтение элементов из файла (последовательный доступ)
Для чтения значений из файла используем объект типа BinaryReader и его методы ReadBoolean(), ReadChar(), ReadDouble(), ReadInt32() и т.д. Создание объекта BinaryReader аналогично созданию объекта класса BinaryWriter. Для последовательного чтения элементов из файла используем цикл:

```
using (FileStream fileStream = File.OpenRead(path))
```

```
using (BinaryReader binaryReader = new BinaryReader(fileStream))
{
    while (binaryReader.PeekChar()!=-1)
    {
        // обработка значений
     }
}
```

Создайте метод

static void ReadFile(string path, uint n = 10, string delimiter = ", ")

который принимает имя файла целых чисел и выводит их на консоль по N чисел в каждой строке (значение по умолчанию для N = 10, N > 0). Для пустого файла выводить сообщение <empty file>. (Используем свойство Length объекта FileStream)

Чтение элементов из файла (произвольный доступ)

В четырёх следующих задачах используем Метод long Seek(long offset, SeekOrigin origin) устанавливает позицию в потоке со смещением на количество байт, указанных в параметре offset. SeekOrigin - перечисление с тремя значениями

SeekOrigin.Begin: начало файла

SeekOrigin.End: конец файла

SeekOrigin.Current: текущая позиция в файле

Смещение может отрицательным, тогда курсор сдвигается назад, если положительное - то вперед. Если мы хотим прочитать из файла целых чисел третье число, мы пишем:

```
using (FileStream fileStream = File.OpenRead(path))
using (BinaryReader binaryReader = new BinaryReader(fileStream))
{
  fileStream.Seek(3 * sizeof(int), SeekOrigin.Begin);
  Console.Write(binaryReader.ReadInt32());
}
```

Дан бинарный файл целых чисел. Обнулить его *минимальный* элемент (считать, что в файле он единственный). Если файл пуст, ничего не делать.**Указание.** Во-первых, следует найти позицию минимального элемента (пользуясь Position).Во вторых, с помощью метода Seek, следует перейти в позицию минимального элемента и записать в файл переменную с нулевым значением.**Тестирование.** В основной программе явно создайте и обработайте четыре файла (обязательно проверить пустой и файл из одного элемента). Для каждого файла: распечатать исходное содержимое, обнулить минимальный элемент, распечатать содержимое файла после изменения. **Указание к тестированию.** Удобно записать имена созданных файлов в массив строк и обработать их в цикле.

Дан бинарный файл целых чисел. Увеличить все его элементы в два раза.**Тестирование.** В основной программе явно создайте и обработайте четыре файла (обязательно проверить пустой и файл из одного элемента).

Дан файл целых чисел (возможно пустой). Инвертировать его (то есть изменить в нём порядок элементов на обратный).**Замечание.** Не забудьте, что использовать вспомогательный файл *запрещается*.**Тестирование.** В основной программе создать и обработать четыре файла (обязательно проверить пустой, файл из одного и двух элементов).

Дан бинарный файл. Удвоить его размер, записав в конец файла все его исходные элементы в *обратном* порядке.**Замечание.** Не забудьте, что использовать вспомогательный файл *запрещается*.**Тестирование.** В основной программе явно создайте и обработайте три файла (обязательно пустой файл, файл из одного целого числа).

Обработка текстовых файлов

```
// Цикл по строкам файла - первый способ
using (var sr = new StreamReader("a.txt"))
{
   string line;
   while ((line = sr.ReadLine()) != null)
      Console.WriteLine(line);
}
//Цикл по строкам файла - второй способ
using (var sr = new StreamReader("a.txt"))
{
   while (!sr.EndOfStream)
      Console.WriteLine(sr.ReadLine());
}
```

Дан текстовый файл. Вывести все символы, содержащиеся в этом файле, и их коды (используйте метод Read()) . Посчитать общее количество символов в выходную переменную метода. Выяснить, какие коды у символов, обозначающих конец строки (ответ укажите в виде комментария). Выяснить код пробела.

Дано целое неотрицательное число *N* и символ *C*. Создать текстовый файл, содержащий *N* строк, первая из которых состоит из одного символа *C*, вторая — из двух, третья — из трёх и т.д. (Пользуемся методами Write() и WriteLine())

Даны два текстовых файла. Дописать содержимое второго файла в конец первого. (Используем StreamWriter, найцдите подходящий конструктор)

Дан текстовый файл. Удалить из него все пустые строки. **Указание.** Все задачи на *изменение* содержимого текстового файла (кроме дополнения) решаются с **использованием временного файла**. Создание временного файла обсуждается на данной странице. Вам **необходимо** написать вспомогательную функцию MakeTempFileName для генерации имени временного файла. Переделайте функцию с данной страницы на язык C# и инициализируйте newFileName по описанному на странице принципу.

Средства System.IO.File для работы с текстовыми файлами

Дан текстовый файл. Определить количество пустых строк в этом файле, используя статический метод ReadLines класса System.IO.File.

Дан csv-файл, содержащий целые числа. Найти сумму чисел в каждой строке файла (решение оформить в виде функции, возвращающей массив целых чисел). Для пустой строки следует возвращать ноль. **Замечание 1.** Каждая строка файла в *формате CSV* (comma-separated values — значения, разделённые запятыми) содержит значения, разделённые запятыми. **Указание 1.** Использовать метод Split для извлечения чисел из строк и метод int.Parse для преобразования их к типу integer для суммирования. **Замечание 2.** Задачу можно решать с использованием явных циклов, либо с использованием методов последовательностей.

Данная задача не требует чтения содержимого файлов.

Создать в текущем каталоге папку text_files, создать там несколько текстовых файлов просто «руками», можете скопировать туда имеющиеся файлы).

Вывести имена файлов каталога text_files (процедура с одним параметром — именем каталога).

Переименовать все текстовые файлы каталога text_files, добавив к именам префикс help-. Это следует выполнить в процедуру с двумя параметрами — именем каталога и префиксом).
Снова вывести имена файлов каталога.
Указания.
Чтобы получить список имён файлов каталога, используйте статический метод System.IO.Directory.GetFiles. Текущему каталогу соответствует имя '.', родительскому — '..'.
Чтобы отделить собственно имя файла от всего пути к файлу (абсолютного или относительного) используйте статический метод System.IO.Path.GetFileName.
Для формирования имени файла с путём по пути и имени используйте System.IO.Path.Combine.
Чтобы переименовать файл, используйте статический метод System.IO.File.Move.
Дано натуральное число *K* и текстовый файл, содержащий слова. Создать текстовый файл и записать в него все слова длины *K* из исходного файла по одному слову в строке. **Указание.** Воспользуйтесь методом System.IO.File.ReadAllText для чтения исходного файла и System.IO.File.WriteAllLines для записи нового файла. **Замечание.** Словом считать набор символов, не содержащий пробелов, знаков препинания и ограниченный пробелами, знаками препинания или началом/концом строки. Если исходный файл не содержит слов длины *K*, то оставить результирующий файл пустым. **Тестирование.** Создать каталог input-files/task-04 и добавить туда несколько входных текстовых файлов, на которых вы проводите тестирование. Убедиться, что соответствующие результирующие файлы содержат то, что нужно.
Дан текстовый файл, в каждой строке которого записано одно или несколько слов. Создать новый текстовый файл из одной строки, в которой через пробел записаны первые слова строк исходного файла, заканчивающиеся на заданную подстроку.

ПРАКТИЧЕСКАЯ РАБОТА №18

инкапсуляция C#
1. ЦЕЛЬ РАБОТЫ
Целью работы является освоить основные навыки работы с объектами и классами, а так же изучить свойства и методы необходимые для создания программ C#.
КРАТКИЕ ТЕОРЕТИЧЕСКИЕ СВЕДЕНИЯ
Класс в C#, как и в других языках программирования, — это пользовательский тип данных (user defined type, UDT), который состоит из данных (часто называемых атрибутами или свойствами) и функциями для выполнения с этими данными различных действий (эти функции обычно называются методами). Классы позволяют группировать в единое целое данные и функциональность, моделируя объекты реального мира. Именно это свойство классов и обеспечивает одно из наиболее важных преимуществ объектно-ориентированных языков программирования.
Столпы объектно-ориентированного программирования
В любом языке программирования C# реализованы три важнейших принципа — «столпа» объектно-ориентированного программирования:
• инкапсуляция: как объекты прячут свое внутреннее устройство;
• наследование: как в этом языке поддерживается повторное использование кода;
• полиморфизм: как в этом языке реализована поддержка выполнения нужного действия в зависимости от типа передаваемого объекта
Инкапсуляция

Первый «столп» объектно-ориентированного программирования — это инкапсуляция. Так называется способность прятать детали реализации объектов от пользователей этих объектов. Например, предположим, что вы создали класс с именем DBReader (для работы с базой данных), в котором определено два главных метода: ОрепО и Close().;

```
// Класс DBReader скрывает за счет инкапсуляции подробности открытия
// и закрытия баз данных
DBReader f
-
new DBReaderO;
f.Open("C:
\
foo.
m
df"};
// Выполняем с базой данных нужные нам действия
f.Close()
;
-
2
–
```

Наш вымышленный класс DBReader инкапсулирует внутренние подробности того, как именно он обнаруживает, загружает, выполняет операции и в конце концов закрывает файл данных. За счет инкапсуляции программирование становится проще: вам нет необходимости беспокоиться об огромном количестве строк кода, которые выполняют свою задачу скрыто от вас. Все, что от вас требуется — создать экземпляр нужного класса и передать ему необходимые сообщения (типа «открыть файл с именем foo.mdf).

Наследование: отношения «быть» и «иметь»

Следующий столп объектно-ориентированного программирования — наследование. Под ним понимается возможность создавать новые определения классов на основе существующих. В сущности, наследование позволяет вам расширять возможности, унаследованные от базового класса, в собственном производном классе.

Отношение «быть»

Как мы помним, на вершине любой иерархии в .NET всегда находится базовый класс Object. В нашей ситуации возможности этого класса вначале дополняются возможностями, привнесенными классом Shape. Речь идет о свойствах, полях, методах и событиях, которые являются общими для всех геометрических фигур (shapes). Класс Hexagon (шестиугольник), производный от Shape, дополняет возможности предыдущих двух базовых классов своими собственными уникальными свойствами.

Когда ваши классы связываются друг с другом отношениями наследования, это означает, что вы устанавливаете между ними отношения типа «быть» (is-a). Такой тип отношений называется также классическим наследованием.

В мире объектно-ориентированного программирования используется еще одна форма повторного использования кода. Эта форма называется включением-делегированием (или отношением «иметь» •—has-a). При ее использовании один класс включает в свой состав другой и открывает внешнему миру часть возможностей этого внутреннего класса.

Полиморфизм: классический и для конкретного случая

Последний, третий столп объектно-ориентированного программирования — это полиморфизм. Можно сказать, что этот термин определяет возможности, заложенные в языке, по интерпретации связанных объектов одинаковым образом. Существует две основные разновидности полиморфизма: классический полиморфизм и полиморфизм «для конкретного случая» (ad hoc). Классический полиморфизм встречается только в тех языках, которые поддерживают классическое наследование (в том числе, конечно, и в C#), При классическом полиморфизме вы можете определить в базовом классе набор членов, которые могут быть замещены в производном классе. При замещении в производных классах членов базового класса эти производные классы будут по-разному реагировать на одни и те же обращения.

Для примера мы вновь обратимся к нашей иерархии геометрических фигур.

Предположим, что в классе Shape (геометрическая фигура) определена функция Draw() — рисование, которая не принимает параметров и ничего не возвращает. Поскольку геометрические фигуры бывают разными и каждый тип фигуры потребуется изображать своим собственным уникальным способом, скорее всего, нам потребуется в производных классах (таких как Hexagon — шестиугольник и Circle — окружность) создать свой собственный метод Draw(), заместив им метод Draw() базового класса

Классический полиморфизм

Классический полиморфизм позволяет определять возможности всех производных классов при создании базового класса. Например, в нашем примере вы можете быть уверены, что метод Draw() в том или ином варианте присутствует в любом классе, производном от Shape. К достоинствам классического полиморфизма можно отнести также и то, что во многих ситуациях вы сможете избежать создания повторяющихся методов для выполнения схожих операций (типа DrawCircle(), DrawRectangle(), DrawHexagon() и т. д.).

Вторая разновидность полиморфизма — полиморфизм для конкретного случая (ad hoc polymorphism). Этот тип полиморфизма позволяет обращаться схожим образом к объектам, не связанным классическим наследованием.

Достигается это очень просто: в каждом из таких объектов должен быть метод с одинаковой сигнатурой (то есть одинаковым именем метода, принимаемыми параметрами и типом возвращаемого значения. В языках, поддерживающих полиморфизм этого типа, применяется технология «позднего связывания» (late binding), когда тип объекта, к которому происходит обращение, становится ясен только в процессе выполнения программы. В зависимости от того, к какому типу мы обращаемся, вызывается нужный метод. Опишем класс как структуру данных, содержащую члены - данные (такие как константы и поля), функциональные члены (методы, свойства, события, индексаторы, операторы, конструкторы и деструкторы) и вложенные типы. Классы также поддерживают наследование.

Простейшее объявление класса выглядит так:

```
// Класс объявленный как public, доступен отовсюду, включая другие сборки
// Можно также написать internal - класс будет доступен только из текущей сборки
public class HelloWorldClass
{
public void HelloWorld()
{
Console.WriteLine("Hello, world");
}
```

}
Можно
также
сделать
класс
абстрактным
: public abstract class AgentFactory
{
public void WriteInfo()
{
Console.WriteLine("I'm agent factory");
}
public abstract Agent CreateAgent();
}
Такие классы могут содержать абстрактные методы (методы, которые не реализованы в текущем классе, но должны быть реализованы в неабстрактных потомках).
Абстрактные классы не могут быть инстанцированы. То есть для использования этого класса, должен быть написан потомок:
public class FootballAgentFactory : AgentFactory
{
public Agent CreateAgent()
{
return new FootballAgent();
}
}
Если же вы наоборот не хотите, чтобы от вашего класса кто-то наследовался, объявите его как sealed: pub
lic sealed class IAmTheCoolest
{
public void WhoAmI()
{
Console.WriteLine("I am cool and I don't want any inheritance");
}
}
//
Это приведет к ошибке компиляции
public class IAmMoreCoolThanTheCoolest : IAmTheCoolest
{
public void I
AmTheCoolest()
{
Console.WriteLine("I'm even cooler that the coolest");
}
}
Как вы уже поняли, после двоеточия указывается класс, от которого наследуется данный. Наследоваться в C# можно только от одного класса, но можно реализовывать несколько интерфейсов: public class Base1
{

```
}
public class Base2
{
}
public interface Interface1
{
}
public interface Interface2
{
}
// Попытка наследования нескольких классов приведет к ошибке
public class MyClass : Base1, Base2
{
}
//
Зато можно реализовывать несколько интерфейсов
public class MyClass2 : Base1, Interface1, Interface2
{
}
```

Если не указан никакой базовый класс, то класс автоматически считается наследником класса System.Object или просто object. В теле класса объявляются его члены. Далее мы опишем все виды членов класса.

Поля

Поле представляет переменную, связанную с данным классом или его экземпляром.

```
public class MyClass
{
// Указывается что это поле доступно из любого класса, даже из другой сборки
// Также можно указать pro
tected
-
поле будет доступно только из классов-наследников
// и private
-
доступно только из текущего класса.
public int a;
// Присваиваем начальное значение в конструкторе
public MyClass()
{
a = 3;
}
}
public
class Application
{
public static void Main()
{
MyClass myClass = new MyClass();
// a = 5
```

```
Console.WriteLine("a = {0}", myClass.a);
myClass.a += 5;
// a = 10
Console.WriteLine("a = {0}", myClass.a);
}
}
```
Также можно обявить поле как readonly - запрещается запись в поле, кроме как при инициалищации посредством инийиализатора или конструктора.
```
public class MyClass
{
// Присваиваем начальное значение в инициализаторе
readonly int a = 1;
// Присваиваем начальное значение в конструкторе
public MyClass()
{
a = 3;
}
public MyMethod()
{
a = 5; //
Ошибка
компиляции
}
}
```
Есть еще модификатор volatile, который запрещает некоторые оптимизации, которые могут привести к непредсказуемым последствиям в многопоточных приложениях.
Модификатор static декларирует принадлежность члена класса всему классу, а не конкретному члену. Доступ к таким членам осуществляется через имя класса, а не его экземпляра.
```
public class MyStaticMemberClass
{
public static int a = 10;
public MyMethod()
{
a = 5;
}
}
public class Application
{
public static void Main()
{
// a = 10
Console.WriteLine("a = {0}", MyStaticMemberClass.a)
;
MyStaticMemberClass.a = 2;
// a = 2
Console.WriteLine("a = {0}", MyStaticMemberClass.a);
}
}
```
Методы

Методы представляют собой действия, которые можно произвести в связи с данным классом.

```
public class MyHelper
{
private int a, b;
public MyHelper( int a, int b )
{
this.a = a;
this.b = b;
}
public int Max()
{
return a > b ? a : b;
}
}
public class Application
{
public static void Main()
{
MyHelper helper = new Helper(2, 10);
// MAX(2, 10) = 10
Console.WriteLine("MAX(2, 10) = {0}", helper.Max());
}
}
```

Возможные модификаторы – уже описанные

public, static, protected, abstract

и еще virtual, override, new и extern. static - то же самое, что и у полей. Указывает. что метод принадлежит классу и а не конкретному экземпляру, работать такой класс может только со статическими членами:

```
public class SampleClass
{
public static int a;
public int b;
public static void DoubleA()
{
a *= 2;
}
public void SetB()
{
```

Важным моментом здесь является инициализация базового класса. Для выхода конструктора базового класса используется ключевое слово base. Вместо долгого формального описания, просто покажем пример: class BaseClass

```
{
int _a;
public int A
{
get
{
```

```
return _a;
}
}
public
BaseClass( int a )
{
_a = a;
}
}
class InheritedClass : BaseClass
{
public InheritedClass() : base(1)
{
}
}
```

Статические конструкторы - инициализируют класс в целом. Никто не знает когда происходит его вызов - гарантирется лишь то, что он будет вызван до создания первого экземпляра класса.

В качестве примера покажем интереснейшую реализацию паттерна Singleton. В разделе "Переменные" был показана стандартная реализация, которую легко перенести на большинство языков. Здесь же мы приведем вариант, использующий особенности C#.

```
public class Singleton
{
private static Singleton _instance;
public static Singleton Instance
{
get
{
return _instance;
}
}
public stati
c Singleton
{
_instance = new Singleton();
}
}
```

Деструкторы

Деструктор - это метод, который вызывается при уничтожении объекта. Так как в .NET CLR используется сборщик мусора, нельзя явно удалить определенный экземпляр.

Удаление происходит когда никакой код уже не может воспользоваться этим экземпляром. Деструкторы определяются как методы с именем, совпадающим с именем класса, предваренным знаком ~ (тильда): public class A

```
{
~A
{
Console.WriteLine("I'm being des
```

```
cructed");
}
}
public class Application
{
public static void Main()
{
A a = new A();
a = null;
}
}
```

Программа выведет

"I'm being destructed"

4. ПОРЯДОК ВЫПОЛНЕНИЯ РАБОТЫ

Порядок выполнения работы:

1. Составить программу, которая продемонстрирует по одному из вариантов, приведенных в следующей таблице. Вводимые значения и результаты вывести на экран дисплея.

Варианты	Описание действий программы
1	Создать класс, внутри которого определить две переменные — целую и строкового типа, а также вложенный класс с аналогичными переменными. Использовать классы для присваивания некоторых значений заданного типа, затем вывести их на экран. Ход программы отобразить с комментариями.
2	Используя свойство наследования создать класс для обработки баз данных двух типов. Создать по классу для каждой из БД. Долю общей функциональности обеих БД поместить в базовый класс, а два других класса сделать его производными. У каждого класса должен быть реализован собственный конструктор, независимый от базового класса. Ход программы отобразить с комментариями.
3	Создать класс, в котором будет 3 числовых и 3 строковых переменных. Затем создать дочерний класс, содержащий все переменные базового класса, а также добавочные обоих типов. При помощи замены методов – изменить 1 из переменных родительского класса в дочернем. Ход программы отобразить с комментариями.
4	Создать базовый класс и подкласс с произвольными переменными. Затем запретить наследование одного из классов и продемонстрировать работу запрета на экране.Ход программы отобразить с комментариями.
5	Создать 2 базовых класса с переменными числового и строкового типов. Затем создать дочерний класс, который будет наследовать числовые целые переменные одного базового класса и строковые второго. Параметры полученного класса вывести на экран. Ход программы отобразить с комментариями.
6	Используя свойство полиморфизма создать базовый класс, в котором определить виртуальный метод, который будет взаимодействовать с 2-мя подклассами в

зависимости от изменения параметров каждого из них. Ход программы отобразить с комментариями.

7

Создать базовый класс и 2 наследующих его подкласса с дополняющими переменными. Затем создать виртуальный метод запрещённый для 1-го подкласса и взаимодействующий со вторым. Ход программы отобразить с комментариями.

8

Создать базовый класс, а затем вложенный в него класс. Затем создать класс, который будет пытаться обратится к вложенному через базовый. Также создать экземпляр вложенного типа. Ход программы отобразить с комментариями.

9

Создать 2 базовых класса и у каждого по 2 дочерних класса с дополнительными переменными. Затем использовать виртуальные методы в каждом из базовых классов, которые будут взаимодействовать с переменными дочерних классов не своего родительского класса. Ход программы отобразить с комментариями.

10

Создать базовый класс и 3 дочерних класса. Сформировать виртуальный метод который будет работать только с 2-мя из дочерних методов. 3-й дочерний класс запретить. Ход программы отобразить комментариями.

2. Ввести программу с клавиатуры с использованием Visual Studio 2005.

3. Отладить программу и запустить на выполнение.

5. СОДЕРЖАНИЕ ОТЧЕТА

В отчете должны быть представлен текст программы, значения вводимых величин и полученные значения выводимых величин.

6. ВОПРОСЫ ДЛЯ САМОКОНТРОЛЯ

1. Что такое классы в C# и для чего они используются?
2. Сформулируйте правила объявления классов.
3. Какие существуют основные «столпы» ООП ?
4. Каков смысл применения свойств инкапсуляции ?
5. Что такое псевдоинкапсуляция ?
6. Можно ли производить наследование от нескольких базовых классов?
7. Сформулируйте суть полиморфизма в C#.

ПРАКТИЧЕСКАЯ РАБОТА №19. НАСЛЕДОВАНИЕ И ПОЛИМОРФИЗМ

2.6. Наследование и полиморфизм Рассмотрим примеры наследования.

Объявим новый класс, наследующий Employee:

```
class Manager : Employee {
// менеджер должен знать количество опционов
private ulong numberOfOpoints;
}
```

Опишем в классе Manager свойство OptionsNumber:

```
// свойство количество опционов
public ulong OptionsNumber {get {
return numberOfOpoints;
}
set {
numberOfOpoints = value;
```

```
}
}
```
В Main создадим представителя класса Manager:
```
Manager m = new Manager();
```
Толку в этом пока нет — нельзя задать значения элементов данных.Изменим тип доступа к элементам данных класса Employee:
```
class Employee {
// название организации
protected static string company_name;
// закрытые данные класса protected EmployeeHelper helper;protected string fullName;
protected short   empID; protected double currPay;
. . .
}
```
В классе Manager опишем конструкторы:
```
// конструктор по умолчанию
public Manager() {
}
// конструктор преобразования
public Manager(short empID) {this.empID = empID;
}
```
Теперь вернемся в функцию Main и изменим создание представителяменеджеров:
```
Manager m = new Manager(102);
```
Значения других элементов класса зададим через свойства, и выведемзначения при помощи DisplayStats:
```
m.Name = "John"; m.Salary = 60000.0;
// информация о менеджере
((Employee)m).DisplayStats();
```
Определим еще один конструктор класса Manager, используя конст-руктор базового класса base:
```
// еще один конструктор класса Manager
public Manager(string name, short id, double pay, ulong p)
: base(name, id, pay) {numberOfOpoints = p;
}
```
Создадим нового представителя класса Manager с использованием но-вого конструктора, конец функции Main:
```
Manager m2 = new Manager("Sam", 103, 40000.0, 20);((Employee)m2).DisplayStats();
```
Переопределим виртуальный метод DisplayStats для класса Manager, используя определенный ранее метод базового класса (base):
```
// выводит сведения о текущем состоянии
public override void DisplayStats() {base.DisplayStats();
Console.WriteLine("Опционов: {0}", numberOfOpoints);
}
```
Теперь вместо строк вида
```
((Employee)m).DisplayStats();
```
можно использовать строки вида
```
m.DisplayStats();
```
Определим еще один производный класс для продавца.Продавец должен знать объем продаж:

```
class Salesman : Employee { private ulong numberOfSales;
    }
```
Определим конструкторы, аналогичные конструкторам Manager:
```
    // конструкторы
    public Salesman() { }
    public Salesman(string name, short id, double pay, ulong p)
    : base(name, id, pay) {numberOfSales = p;
    }
```
Опишем виртуальный метод DisplayStats для продавца:
```
    // выводит сведения о текущем состоянии
public override void DisplayStats() {base.DisplayStats();
    Console.WriteLine("Продаж: {0}", numberOfSales);
    }
```

В функции Main создаем нового продавца и выводим его данные:
```
Salesman s = new Salesman("Stan", 201, 20000, 10);s.DisplayStats();
```
Создадим еще один класс, производный от класса продавца — прода- вец на неполный рабочий день (part-time salesman). Мы хотим также, что- бы от нового класса нельзя было далее продолжать наследование. В этом случае класс определяется со словом sealed:
```
// класс, заканчивающий цепочку наследования
sealed class POINTSalesman : Salesman {
}
```
Данный класс не может выступать в качестве базового.Определим конструкторы нового класса:
```
// конструкторы
public POINTSalesman() { }
public POINTSalesman(string name, short id, double pay, ulong
p)
: base(name, id, pay, p) {
}
```
В функции Main создадим представителя и выведем его данные:
```
POINTSalesman points = new POINTSalesman("WhoAmI", 301,
10000, 8);
points.DisplayStats();
```

2.7. Пространства имен

Создадим проект консольного приложения C#.
Название проекта — SharpNS.

Пространство имен — это логическая структура для организациииимен. Они позволяют сгруппировать определяемые типы данных.
Определим три пространства имен:
```
namespace baseshapes {
}
namespace shapes {
}
namespace shapes {
}
```

Обратим внимание, что два пространства имен названы одинаково.В первом пространстве имен располагаем описание класса figure:

```
namespace baseshapes {class figure {
}
}
```

Во втором пространстве имен определим класс circle:

```
namespace shapes { using baseshapes;
class circle : figure {
}
}
```

В третьем пространстве имен определим класс square.
Опишем в классе figure абстрактный метод:

```
class figure {
virtual public void draw() {}
}
```

В классах circle и square опишем реализацию метода draw, используямодификатор override. Пусть методы выводят в консоль названия классов.
В функции Main можно создать представителя класса circle:

```
class Program {
static void Main(string[] args) { shapes.circle c = new shapes.circle();c.draw();
}
}
```

При помощи using то же самое сделать будет проще. Указываем ис-пользование пространства имен в начале модуля:

```
using shapes;
```

В методе Main можно не использовать пространство имен shapes:

```
class Program {
static void Main(string[] args) { shapes.circle c = new shapes.circle();c.draw();
square s = new square();s.draw();
}
}
```

2.8. Контрольные вопросы и упражнения

1. Чем структура отличается от класса?
2. Что называется статическим конструктором?
3. Для чего нужен статический конструктор?
4. Как определяются методы класса?
5. Как определяются свойства класса?
6. Чем свойство отличается от метода?
7. Как определяется наследование классов?
8. Как переопределяется виртуальный метод класса?

ПРАКТИЧЕСКАЯ РАБОТА № 20. Абстрактные классы и интерфейсы

Цели:

- абстрактные классы и интерфейсы.Задачи:
- абстрактные классы;
- абстрактные методы;

- наследование абстрактных классов;
- наследование интерфейсов.Опорные документы:

5.1. Базовый класс

Создадим проект консольного приложения C#.
Название проекта — SharpShapes.
Наша первая цель — воспроизвести иерархию классов геометриче-ских фигур Shape — Circle, Square.
Класс Shape. Опишем следующие элементы класса:

- защищенное поле name типа string, значение по умолчанию "figure",
- конструктор по умолчанию,
- конструктор с параметром типа string, задает поле name,
- не виртуальный метод Draw, выводит в консоль "Shape имя"
- свойство Name.

После определения класса создадим в методе Main представителя, вы-зовем метод Draw. Убедимся, что в консоль выводится название класса и имя объекта.
Замечание по формированию конструктора. В C# в конструкторе есть список инициализации, как и в C++, однако в нем можно вызывать только конструкторы. Конструктор базового класса вызывается при помощи клю- чевого слова base, другой конструктор этого же класса вызывается при по- мощи ключевого слова this.
Вы можете попробовать создать конструктор с параметром, задаю- щим поле name, следующим образом:

```
class Shape {
public Shape(string n) : name(n) {
}
}
```

При этом компилятор будет сообщать об ошибке.Так нельзя.
С другой стороны, вы можете определить конструктор с двумя пара- метрами. Для этого добавьте в класс закрытое поле ID целого типа, значе- ние по умолчанию нулевое.
Теперь можно задать два следующих конструктора:

```
public Shape(string name) {this.name = name;
}
public Shape(string name, int id) : this(name) {ID = id;
}
```

Здесь this(name) вызывает конструктор с одним параметром.

5.2. Абстрактный класс

Сделаем класс Shape абстрактным.
Для этого к описанию класса добавим модификатор abstract:

```
abstract class Shape {
```

Убедимся, что создать представителя класса теперь нельзя, поэтому удалим из Main создание представителя. Абстрактный класс может слу- жить только в качестве базового класса. Поэтому далее описываем два производных класса Circle и Square. Методы Draw этих классов выводят в консоль "Circle имя" и "Square имя" соответственно.
Заметим, что методы Draw не могут иметь модификатор override, этот модификатор допустим, только если соответствующий метод базового класса имеет модификатор virtual, abstract или override.
В Main создаем представителей классов и вызываем их методы Draw.
Убеждаемся, что методы выводят правильное название класса:

Circle circleSquare square
Теперь попробуем следующий код:
Shape fc = new Circle("circle");fc.Draw();
Shape fs = new Square("square");fs.Draw();
Убедимся, что в консоль выводятся строки:
Circle circleSquare squareShape circle Shape square
Этого и следовало ожидать. Теперь включим полиморфизм.
В метод Draw базового класса добавим модификатор virtual, в методы Draw производных классов добавим модификатор override.
Убедимся, что выводятся строки:
Circle circleSquare squareCircle circleSquare square
Таким образом, иерархия классов фигур реализована.

5.3. Абстрактный метод

В абстрактном классе можно определить абстрактный метод.
Пробуем приписать модификатор abstract к методу Draw базовогокласса Shape.
Компилятор сообщит об ошибке.
Во-первых, метод не должен иметь реализации:
public abstract virtual void Draw();
Компилятор все равно не допускает этот код.
Во-вторых, абстрактный метод виртуальный по определению, поэто- му нужно удалить модификатор virtual, — модификаторы abstract и virtual взаимоисключающие, можно использовать либо тот, либо другой.
После удаления модификатор virtual программа снова заработает.
Интересно также узнать, можно ли определить абстрактный метод в не абстрактном базовом классе. Пробуем удалить модификатор abstract у класса Shape:
class Shape {
Убедимся, что компилятор сопротивляется.
Нельзя определять абстрактный метод в не абстрактном классе.

5.4. Интерфейсы

Класс C# может наследовать только один базовый класс, множествен- ное наследование классов исключено. Однако разрешено множественное наследование интерфейсов.
Интерфейс определяется иначе, чем абстрактный класс.
Отличие проявляется в том, что методы интерфейса являются абст- рактными (и виртуальными) по определению, поэтому никакие модифика- торы в описании методов недопустимы, даже public. Кроме методов, ин- терфейс может содержать только свойства, события и индексаторы.
Первый интерфейс, который мы определим, это IDrawable.
Этот интерфейс определяет только метод Draw:
interface IDrawable {void Draw();
}
Описание интерфейса, как видим, максимально упрощено. Теперь описываем наследование интерфейса классом Shape:
abstract class Shape : IDrawable {
Удивительно, но компилятор не сопротивляется нашим действиям.
Попробуйте закомментировать метод Draw класса Shape. Для чистотыэксперимента нужно также закомментировать описание производныхклассов и код метода Main.
Теперь компилятор сопротивляется.

Если в интерфейсе есть метод, в производном абстрактном классе ондолжен быть абстрактным или иметь реализацию.
Абстрактным метод Draw класса Shape является сейчас.Пусть производные классы описаны.
Попробуем приписать реализацию методу Draw класса Shape:
public virtual void Draw() { Console.WriteLine("Shape {0}", name);
}
Компилятор допускает такую форму метода.
Таким образом, абстрактный класс, наследующий интерфейс, может определять либо абстрактный, либо виртуальный метод. Модификатор override в этом случае в классе Shape недопустим.
Вернем назад абстрактный метод Draw класса Shape:
public abstract void Draw();
Опишем интерфейс IPointy (угловатый):
interface IPointy {
// абстрактный элемент интерфейса
int Points();
}
Описываем наследование этого интерфейса классом Square, потому что квадрат имеет четыре угла:
class Square : Shape, IPointy {
Теперь мы обязаны описать метод Points в классе Square, метод воз- вращает значение 4. Реализация должна отличаться от метода интерфейса обязательным наличием модификатора public.
Экспериментируем в Main:
Console.WriteLine("points = {0}", s.Points()); Console.WriteLine("points = {0}", fs.Points());
Выясняем, что вторая строка недопустима. Объект s имеет тип Square, объект fs имеет тип Shape.

5.5. Определение наличия интерфейса

Как узнать, что некоторый объект реализует некоторый интерфейс?Есть несколько способов.
Способ 1. Пробуем привести объект к типу интерфейса. Если приве- дение невозможно, возникает исключение InvalidCastException, поэтомумы должны использовать блоки try и catch:
IPointy ip;try {
} catch (InvalidCastException) { Console.WriteLine("No IPointy");
}
Это полигон для испытания.
В блок try вписываем сначала приведение объекта s к типу IPointy и пробуем присвоить это переменной ip типа интерфейса:
ip = (IPointy)s;
Console.WriteLine("points = {0}", ip.Points());
Убедимся, что этот фокус удается. Теперь попробуем сделать то же самое для объекта fs, затем для объекта c класса Circle. Результат тестиро- вания записываем в отчет.
Способ 2. Используем ключевое слово as. Полигон для этого испыта- ния выглядит следующим образом:
ip = s as IPointy;if (ip == null) {

```
Console.WriteLine("No as IPointy");
} else {
Console.WriteLine("points = {0}", ip.Points());
}
```
Тестирование заключается в использовании вместо переменной s вприведенном фрагменте кода всех других переменных.
Способ 3. Используем ключевое слово is в условии. Пример кода дляобъекта s имеет следующий вид:
```
if (s is IPointy) {
Console.WriteLine("points={0}",((IPointy)s).Points());
} else {
Console.WriteLine("No is IPointy");
}
```
Обратим внимание, что объект в некоторых случаях требует приведе-ния к типу IPointy. В некоторых случаях приведение не обязательно.
Опишем еще один интерфейс:
```
public interface IDraw3D {void Draw();
}
```
Пусть класс Square наследует два интерфейса:
```
class Square : Shape, IPointy, IDraw3D
```
Поскольку Square переопределяет метод Draw, следующий код приво-дит к двусмысленности:
```
if (s is IDraw3D) { ((IDraw3D)s).Draw();
} else {
Console.WriteLine("No IDraw3D");
}
```
Здесь используется имеющийся метод Draw класса Square.
В этом случае нужно переопределить метод интерфейса IDraw3D сиспользованием квалификатора следующим образом:
```
// IDraw3D
void IDraw3D.Draw() { Console.WriteLine("Square3D {0}", name);
}
```
Теперь все выводится правильно.

5.6. Интерфейсы как параметры методов

Указатели на интерфейсы могут быть использованы как параметры методов. Опишем метод для рисования трехмерных объектов, параметр метода — указатель на интерфейс IDraw3D:
```
class Program {
public static void DrawShapeIn3D(IDraw3D itf3d) { Console.WriteLine("***
DrawShapeIn3D ***"); itf3d.Draw();
}
```

Теперь в Main можно вызвать эту функцию для тех объектов, которыеподдерживают интерфейс IDraw3D, например:
```
if (s is IDraw3D) { ((IDraw3D)s).Draw(); DrawShapeIn3D((IDraw3D)s);
} else {
Console.WriteLine("No IDraw3D");
}
```

5.7. Иерархия интерфейсов

Создадим проект консольного приложения C#.
Название проекта — SharpInterfaces.
Интерфейсы могут образовывать иерархии. Если класс наследует один из интерфейсов иерархии, то он должен переопределить все методы всех интерфейсов вверх по иерархии.
Опишем иерархию следующих интерфейсов:

```
// иерархия интерфейсов
interface IDraw {void Draw();
}
interface IDraw2 : IDraw {void DrawToPrinter();
}
interface IDraw3 : IDraw2 {void DrawToMetafile();
}
```

Опишем класс, который наследует все интерфейсы иерархии:

```
class SuperImage : IDraw3 {
}
```

Класс SuperImage должен определять методы всех интерфейсов.Описываем все методы.
Пусть методы выводят следующие строки:
IDraw.Draw IDraw2.DrawToPrinter IDraw3.DrawToMetafile
В методе Main создаем представителя SuperImage, и описываем поли-гон для испытаний:

```
SuperImage si = new SuperImage();try {
}
catch (InvalidCastException) { Console.WriteLine("No Interface");
}
```

В блоке try последовательно подставляем следующие фрагменты кода.Фрагмент 1:

Фрагмент 2:

Фрагмент 2:

```
// получим ссылку на интерфейс IDrawIDraw idr = (IDraw)si;
idr.Draw();
```

```
// получим ссылку на интерфейс IDraw2IDraw2 idr2 = (IDraw2)si; idr2.Draw();
idr2.DrawToPrinter();
```

```
// не хотим использовать ссылку на интерфейс IDraw3((IDraw3)si).Draw();
((IDraw3)si).DrawToPrinter(); ((IDraw3)si).DrawToMetafile();
```

5.8. Контрольные вопросы и упражнения

1. Что означают this() и base() в списке инициализации конструктора?
2. Как описать абстрактный класс?
3. Как описать абстрактный метод?
4. Какой модификатор может иметь метод базового класса для того,чтобы метод производного класса мог иметь модификатор override?

5. Если абстрактный класс наследует интерфейс, какое определениеметода интерфейса может быть в абстрактном классе?
6. Как описывается интерфейс?
7. Какие элементы может содержать интерфейс?
8. Сколько базовых классов и интерфейсов может иметь класс?
9. Как определить наличие интерфейса у объекта?
10. Как определяются методы с одинаковым названием, но принадле-жащие разным интерфейсам?

ПРАКТИЧЕСКАЯ РАБОТА № 21. Интерфейсы и коллекции C#

Цели:

- реализация системных интерфейсов.Задачи:
- интерфейсы IEnumerable и IEnumerator;
- интерфейс ICloneable;
- интерфейс IComparable;
- интерфейс IComparer;
- класс ArrayList;

Опорные документы:

6.1. Интерфейсы перечисления

Создадим проект консольного приложения C#.

Название проекта — SharpNumerable.

Microsoft .NET объединяет в библиотеке базовых классов все лучшее, что накоплено в программировании за последние десятилетия. При изуче- нии библиотеки базовых классов можно обнаружить множество классов- заготовок, использующих одни и те же стандартные интерфейсы. Следует научиться использовать эти интерфейсы, чтобы создавать свои классы.

Опишем класс Car (автомобиль):

```
class Car {
private string name = "noname";public Car() { }
public Car(string name) {this.name = name;
}
public string Name {
get { return name; }
}
}
```

Опишем класс Cars (свалка автомобилей):

```
class Cars {
private Car[] cars;public Cars() {
cars = new Car[4];
cars[0] = new Car("alpha"); cars[1] = new Car("porshe"); cars[2] = new Car("nissan");
cars[3] = new Car("ГАЗ-13");
}
}
```

Было бы очень удобно использовать цикл foreach/in для просмотра свалки. А для этого нужно всего лишь описать интерфейс IEnumerable (пе- речисляемый).

Прежде необходимо удалить последнее слово Generic в using:

using System.Collections.Generic;
Должно получиться:
using System.Collections;
Описываем наследование класса IEnumerable:
class Cars : IEnumerable {
Тип IEnumerable имеет один метод GetEnumerator, возвращающий тип IEnumerator. И мы должны описать этот метод в классе Cars:

```
// IEnumerable
public IEnumerator GetEnumerator() {
for (int i = 0; i < cars.Length; i++) {
// многократный возврат
yield return cars[i];
}
}
```

Переходим в метод Main.
Создаем свалку автомобилей и просматриваем ее:

```
Cars carlot = new Cars(); foreach (Car c in carlot) {
Console.WriteLine(c.Name);
}
```

При наследовании интерфейса IEnumerator нужно описать также его элементы: void Reset — перемещение к первому элементу (метод), bool MoveNext — перемещение к следующему элементу (метод), object Current
— возвращает текущий элемент (свойство).
Описываем наследование IEnumerator:
class Cars : IEnumerable, IEnumerator {
Описываем индекс текущего элемента (в классе Cars):
private int current = -1;
Описываем реализацию метода Reset (в классе Cars):

```
// IEnumerator
public void Reset() {current = -1;
}
```

Описываем реализацию метода MoveNext (в классе Cars):

```
// IEnumerator
public bool MoveNext() {
if (current < cars.Length - 1) {current++;
return true;
} else {
return false;
}
}
```

Описываем реализацию свойства Current (в классе Cars):

```
// IEnumerator
public object Current {get {
return cars[current];
}
}
```

В методе Main вызываем новые методы (после цикла foreach/in):
carlot.Reset(); carlot.MoveNext();

```
Console.WriteLine(((Car)carlot.Current).Name);
```
Кроме того, описав методы и свойства IEnumerator, мы можем значи-тельно упростить метод GetEnumerator класса Cars:
```
// IEnumerable
public IEnumerator GetEnumerator() {return (IEnumerator)this;
/*/
for (int i = 0; i < cars.Length; i++) {
// многократный возврат
yield return cars[i];
}
/*/
}
```

6.2. Внутренний перечислитель

Создадим проект консольного приложения C#.
Название проекта — SharpNuminside.
Снова опишем класс Car:
```
class Car {
private string name = "noname";public Car() { }
public Car(string name) {this.name = name;
}
public string Name {
get { return name; }
}
}
```
Снова опишем класс Cars.
При этом создадим внутренний класс CarsEnumerator, как рекоменду-ет MSDN 2005.
Изменять строки using не требуется.Описание класса Cars:
```
class Cars : System.Collections.IEnumerable {private Car[] cars;
} // конец класса Cars
```
Конструктор класса Cars не изменился:
```
public Cars() {
cars = new Car[4];
cars[0] = new Car("alpha"); cars[1] = new Car("porshe");cars[2] = new Car("nissan");
cars[3] = new Car("ГАЗ-13");
}
```
Объявим закрытый класс CarsEnumerator, который будет выполнятьроль перечислителя. Класс находится внутри класса Cars:
```
private class CarsEnumerator : System.Collections.IEnumerator {private int current = -1;
private Cars c;
} // конец класса CarsEnumerator
```
Конструктор класса CarsEnumerator принимает ссылку на внешнийкласс:
```
public CarsEnumerator(Cars c) {this.c = c;
}
```
Остальная часть класса CarsEnumerator описывает необходимые эле-менты интерфейса IEnumerator:
```
public void Reset() {current = -1;
}
```

```
public bool MoveNext() {
if (current < c.cars.Length - 1) {current++;
return true;
} else {
return false;
}
}
public object Current {
get { return c.cars[current]; }
}
} // конец класса CarsEnumerator
```

Наконец, описываем метод GetEnumerator класса Cars:

```
public System.Collections.IEnumerator GetEnumerator() { return new CarsEnumerator(this);
}
} // конец класса Cars
```

Убедимся, что проект не содержит ошибок.

В методе Main создаем свалку автомобилей carlot и описываем про-смотр всех объектов:

```
static void Main(string[] args) { Cars carlot = new Cars(); foreach (object o in carlot) {
Console.WriteLine(((Car)o).Name);
}
}
```

6.3. Клонируемые объекты

Создадим проект консольного приложения C#.

Название проекта — SharpClonable.

Опишем класс Point (точка на плоскости):

```
class Point {
public int x, y; public Point() { }
public Point(int x, int y) {this.x = x;
this.y = y;
}
public override string ToString() {
return "(" + x.ToString() + "," + y.ToString() + ")";
}
}
```

В методе Main испытываем, как он работает:

```
static void Main(string[] args) {Point p1 = new Point(2, 3);
Console.WriteLine("1: " + p1.ToString());
}
```

Теперь попробуем создать копию объекта Point:

```
Point p2 = p1;p2.x = 3;
Console.WriteLine("2: " + p2.ToString());Console.WriteLine("1: " + p1.ToString());
```

Поскольку объекты класса являются ссылочными типами, при ис-пользовании оператора = создается новая ссылка на тот же объект.

При этом используется метод MemberwizeClone по умолчанию.

Чтобы получить копию объекта вместо копии ссылки, нужно описатьспособ создания копии, используя интерфейс ICloneable.

Описываем класс Point2, наследующий интерфейс ICloneable:

```
class Point2 : ICloneable {public int x, y;
public Point2() { }
public Point2(int x, int y) {this.x = x;
this.y = y;
}
public override string ToString() {
return "(" + x.ToString() + "," + y.ToString() + ")";
}
}
```

Переопределяем единственный метод Clone интерфейса ICloneable:

```
// ICloneable
public object Clone() {
return new Point2(this.x, this.y);
}
```

Испытываем новый класс в методе Main:

```
Point2 p3 = new Point2(6, 6); Point2 p4 = (Point2)p3.Clone();p4.x = 3;
Console.WriteLine("4: " + p4.ToString());Console.WriteLine("3: " + p3.ToString());
```

Убедимся, что в консоль выводится:

```
1: (2,3)
2: (3,3)
1: (3,3)
4: (3,6)
3: (6,6)
```

6.4. Сравниваемые объекты

Создадим проект консольного приложения C#.

Название проекта — SharpComparable.

Снова опишем класс Car:

```
class Car {
private string name = "noname";public Car() { }
public Car(string name) {this.name = name;
}
public string Name {
get { return name; }
}
}
```

В методе Main создаем и выводим список автомобилей:

```
static void Main(string[] args) {Car[] cars = new Car[4]; cars[0] = new Car("3");
cars[1] = new Car("2");
cars[2] = new Car("1");
cars[3] = new Car("4"); foreach (object c in cars) {
Console.WriteLine(((Car)c).Name);
}
}
```

Пусть нам теперь требуется упорядочить список при помощи метода System.Array.Sort. Чтобы это было возможно сделать, объекты массивадолжны поддерживать интерфейс IComparable (сравниваемые).
Добавляем наследование интерфейса IComparable в класс Car:
class Car : IComparable {
Описываем в классе Car, как сравнивать объекты класса Car.
Мы используем сравнение объектов на основе имени:
// IComparable
int IComparable.CompareTo(object o) { return name.CompareTo(((Car)o).name);
}
Теперь в методе Main мы можем отсортировать массив:
Array.Sort(cars); Console.WriteLine("Sorted:");foreach (object c in cars) {
Console.WriteLine(((Car)c).Name);
}

Создадим проект консольного приложения C#.
Название проекта — SharpComparer.
Снова опишем класс Car:
class Car {
private string name = "noname";public Car() { }
public Car(string name) {this.name = name;
}
public string Name {
get { return name; }
}
}
Есть еще один интерфейс, используемый для сравнения объектов — интерфейс IComparer. Он предназначен для сравнивания каких-либо зна- чений двух объектов, и состоит из метода Compare.
Описываем класс сравнивателя:
class SortByName : System.Collections.IComparer {public SortByName() { }
// IComparer
int System.Collections.IComparer.Compare(object a, object b) { return String.Compare(((Car)a).Name, ((Car)b).Name);
}
}
Заметим, что данный класс ориентирован на сравнение типов Car.
В методе Main создаем массив автомобилей:
Car[] cars = new Car[4];cars[0] = new Car("4");
cars[1] = new Car("1");
cars[2] = new Car("3");
cars[3] = new Car("2");
и выводим список:
foreach (Car c in cars) { Console.WriteLine(c.Name);
}
Далее сортируем массив и выводим его:
Array.Sort(cars, new SortByName()); Console.WriteLine("Sorted:"); foreach (Car c in cars) {

Console.WriteLine(c.Name);
}
Недостаток описанного подхода заключается в том, что при проекти- ровании класса приходится учитывать внешние по отношению к нему ме- тоды. Было бы удобнее вместо этого использовать метод класса.
Метод Array.Sort имеет специально перегруженный вариант для этого случая. В классе Car нужно объявить статический метод, возвращающий IComparer:
public static System.Collections.IComparer SortByName {get {
return (System.Collections.IComparer)new SortByName();
}
}
Здесь new SortByName — представитель класса SortByName.
В методе Main теперь можно заменить вызов Sort:
//Array.Sort(cars, new SortByName());Array.Sort(cars, Car.SortByName);

6.5. Коллекции C#

Создадим проект консольного приложения C#.
Название проекта — SharpCollection.
Снова опишем класс Car:
class Car {
private string name = "noname";public Car() { }
public Car(string name) {this.name = name;
}
public string Name {
get { return name; }
}
}
Встроенные в библиотеку базовых классов типы пространства имен System.Collections предназначены для работы с наборами элементов, и яв- ляются контейнерными классами.
Интересны следующие интерфейсы:
Класс ICollection — определяет общие характеристики коллекций.Класс IDictionary — представляет объект парами имя-значение.
Класс IList — добавление, удаление и индексирование элементов. Кроме того, есть несколько готовых классов-коллекций, например:Класс ArrayList — динамический массив объектов.
Класс Queue — стандартная очередь.
Класс SortedList — аналогичен словарю, но является индексируемым.Класс Stack — стандартный стек.
Рассмотрим применение класса ArrayList. Сначала исправим using:
using System.Collections;
Описываем новый класс Cars:
class Cars : IEnumerable { private ArrayList cars;public Cars() {
cars = new ArrayList();
}
}
Добавим в класс Cars метод для добавления нового объекта:
public void Add(Car c) {cars.Add(c);
}

Следующее свойство возвращает количество объектов:

```
public int Count {get {
return cars.Count;
}
}
```

Следующий метод удаляет все объекты:

```
public void Clear() {cars.Clear();
}
```

Следующий метод удаляет элемент коллекции по его индексу:

```
public void RemoveAt(int index) {cars.RemoveAt(index);
}
```

Следующий метод проверяет наличие элемента в коллекции:

```
public bool Contains(Car c) {return cars.Contains(c);
}
```

Последний метод — метод перечисления интерфейса IEnumerable:

```
// IEnumerable
public IEnumerator GetEnumerator() {return cars.GetEnumerator();
}
```

Тестируем класс в методе Main.

Сначала создаем представителя класса Cars и добавляем элементы:

```
static void Main(string[] args) {Cars cars = new Cars(); cars.Add(new Car("Alpha"));
cars.Add(new Car("RR"));
cars.Add(new Car("117")); cars.Add(new Car("Nissan"));
}
```

Затем выводим список:

```
Console.WriteLine("We have {0} cars:", cars.Count);foreach (Car c in cars) {
Console.WriteLine(c.Name);
}
```

Удаляем третий элемент:

```
cars.RemoveAt(3);
Console.WriteLine("We have {0} cars:", cars.Count);foreach (Car c in cars) {
Console.WriteLine(c.Name);
}
```

Добавляем новый элемент:

```
Car a = new Car("Audi100");cars.Add(a);
```

Проверяем наличие нового элемента:

```
if (cars.Contains(a)) { Console.WriteLine(a.Name + " is present");
}
```

Наконец, удаляем все:

```
cars.Clear();
Console.WriteLine("We have {0} cars.", cars.Count);
```

Интересный вопрос возникает — нельзя просто воспроизвести Cars отArrayList? Можно, но тогда в класс можно будет добавлять произвольные объекты. В конце метода Main продемонстрируем использование ArrayList в таком качестве:

```
ArrayList ar = new ArrayList();ar.Add(cars);
ar.Add(new Car("abc"));
ar.Add("Hi");ar.Add(33);
foreach (object o in ar) { Console.WriteLine(o.ToString());
```

}

6.6. Контрольные вопросы и упражнения

1. Перечислите методы интерфейса IEnumerable.
2. Перечислите методы интерфейса IEnumerator.
3. Перечислите методы интерфейса ICloneable.
4. Перечислите методы интерфейса IComparable.
5. Перечислите методы интерфейса IComparer.
6. Перечислите методы класса ArrayList.

Практическая работа №24 «Создание простых программ в среде Visual Studio» Составить программу с использованием визуальных элементов для ввода-вывода данных, кнопки и т.п. Оформить форму и все ее элементы, используя цветовую гамму

1. Дана сторона квадрата a. Найти его периметр P = 4·a.
2. Дана сторона квадрата a. Найти его площадь S = a2.
3. Даны стороны прямоугольника a и b. Найти его площадь S = a·b и периметр P = 2·(a + b).
4. Дан диаметр окружности d. Найти ее длину L = π·d.
5. Дана длина ребра куба a. Найти объем куба V = a3 и площадь его поверхности S = 6·a2.
6. Даны длины ребер a, b, c прямоугольного параллелепипеда. Найти его объем V = a·b·c и площадь поверхности S = 2·(a·b + b·c + a·c).
7. Найти длину окружности L и площадь круга S заданного радиуса R: L = 2·π·R, S = π·R2.
8. Даны два числа a и b. Найти их среднее арифметическое.
9. Даны два неотрицательных числа a и b. Найти их среднее геометрическое, то есть квадратный корень из их произведения.
10. Даны два ненулевых числа. Найти сумму, разность, произведение и частное их квадратов.
11. Даны два ненулевых числа. Найти сумму, разность, произведение и частное их модулей.
12. Даны катеты прямоугольного треугольника a и b. Найти его гипотенузу c и периметр P.
13. Даны два круга с общим центром и радиусами R1 и R2 (R1 > R2). Найти площади этих кругов S1 и S2, а также площадь S3 кольца, внешний радиус которого равен R1, а внутренний радиус равен R2: S1 = π·(R1)2, S2 = π·(R2)2, S3 = S1 – S2.
14. Дана длина L окружности. Найти ее радиус R и площадь S круга, ограниченного этой окружностью, учитывая, что L = 2·π·R, S = π·R2.
15. Дана площадь S круга. Найти его диаметр D и длину L окружности, ограничивающей этот круг, учитывая, что L = π·D, S = π·D2/4.
16. Найти расстояние между двумя точками с заданными координатами x1 и x2 на числовой оси: |x2 – x1|.
17. Даны три точки A, B, C на числовой оси. Найти длины отрезков AC и BC и их сумму.
18. Даны три точки A, B, C на числовой оси. Точка C расположена между точками A и B. Найти произведение длин отрезков AC и BC.
19. Даны координаты двух противоположных вершин прямоугольника: (x1, y1), (x2, y2). Стороны прямоугольника параллельны осям координат. Найти периметр и площадь данного прямоугольника.
20. Найти расстояние между двумя точками с заданными координатами (x1, y1), (x2, y2) на плоскости.
21. Даны координаты трех вершин треугольника(x1, y1), (x2, y2), (x3, y3). Найти его периметр, используя формулу для расстояния между двумя точками на плоскости.

22. Поменять местами содержимое переменных A и B и вывести новые значения A и B.
23. Даны переменные A, B, C. Изменить их значения, переместив содержимое A в B, B — в C, C — в A, и вывести новые значения переменных A, B, C.
24. Даны переменные A, B, C. Изменить их значения, переместив содержимое A в C, C — в B, B — в A, и вывести новые значения переменных A, B, C.
25. Найти значение функции $y = 3 \cdot x^6 - 6 \cdot x^2 - 7$ при данном значении x.

Практическая работа №2 «Создание программ и работа с текстовыми файлами»

Составить программу с использованием визуальных элементов и операторов языка программирования C#. Оформить форму и все ее элементы.

1. Дано целое число k и файл, содержащий неотрицательные целые числа. Вывести k-ый элемент файла (элементы нумеруются от 1). Если такой элемент отсутствует, то вывести –1.
2. Дан файл целых чисел, содержащий не менее четырех элементов. Вывести первый, второй, предпоследний и последний элементы данного файла.
3. Даны имена двух файлов вещественных чисел. Известно, что первый из них существует и является непустым, а второй в текущем каталоге отсутствует. Создать отсутствующий файл и записать в него начальный и конечный элементы существующего файла (в указанном порядке).
4. Даны имена двух файлов вещественных чисел. Известно, что один из них (не обязательно первый) существует и является непустым, а другой в текущем каталоге отсутствует. Создать отсутствующий файл и записать в него конечный и начальный элементы существующего файла (в указанном порядке).
5. Дан файл целых чисел. Создать новый файл, содержащий те же элементы, что и исходный файл, но в обратном порядке.
6. Дан файл вещественных чисел. Создать два новых файла, первый из которых содержит элементы исходного файла с нечетными номерами (1, 3, …), а второй — с четными (2, 4, …).
7. Дан файл целых чисел. Создать два новых файла, первый из которых содержит четные числа из исходного файла, а второй — нечетные (в том же порядке). Если четные или нечетные числа в исходном файле отсутствуют, то соответствующий результирующий файл оставить пустым.
8. Дан файл целых чисел. Создать два новых файла, первый из которых содержит положительные числа из исходного файла (в обратном порядке), а второй — отрицательные (также в обратном порядке). Если положительные или отрицательные числа в исходном файле отсутствуют, то соответствующий результирующий файл оставить пустым.
9. Дан файл вещественных чисел. Найти среднее арифметическое его элементов.
10. Дан файл вещественных чисел. Найти сумму его элементов с четными номерами.
11. Дан файл целых чисел. Найти количество содержащихся в нем серий (то есть наборов последовательно расположенных одинаковых элементов).
12. Дан файл вещественных чисел. Найти его первый локальный минимум (локальным минимумом называется элемент, который меньше своих соседей).
13. Дан файл вещественных чисел. Найти его последний локальный максимум (локальным максимумом называется элемент, который больше своих соседей).
14. Дан файл вещественных чисел. Найти общее количество его локальных экстремумов, то есть локальных минимумов и локальных максимумов.

15. Дан файл вещественных чисел. Создать файл целых чисел, содержащий номера всех локальных максимумов исходного файла в порядке возрастания.
16. Дан файл вещественных чисел. Создать файл целых чисел, содержащий номера всех локальных экстремумов исходного файла в порядке убывания.
17. Дан файл вещественных чисел. Заменить в нем все элементы на их квадраты.
18. Дан файл вещественных чисел. Заменить в файле каждый элемент, кроме начального и конечного, на его среднее арифметическое с предыдущим и последующим элементом.
19. Дан файл целых чисел, содержащий более 50 элементов. Уменьшить его размер до 50 элементов, удалив из файла необходимое количество конечных элементов.
20. Дан файл целых чисел, содержащий четное количество элементов. Удалить из данного файла вторую половину элементов.
21. Дан файл целых чисел, содержащий более 50 элементов. Уменьшить его размер до 50 элементов, удалив из файла необходимое количество начальных элементов.
22. Дан файл целых чисел. Удалить из него все элементы с четными номерами.
23. Дан файл целых чисел. Удалить из него все отрицательные числа.
24. Дан файл целых чисел, содержащий менее 50 элементов. Увеличить его размер до 50 элементов, записав в начало файла необходимое количество нулей.
25. Дан файл вещественных чисел. Поменять в нем местами минимальный и максимальный элементы.

Практическая работа №31 «Использование элементов управления с поддержкой редактирования текста и методов на C#»

Составить программу с использованием элементов управления с поддержкой редактирования текста и операторов языка программирования C#. Оформить форму и все ее элементы.

1. Дано целое число N (> 0). Сформировать и вывести целочисленный массив размера N, содержащий N первых положительных нечетных чисел: 1, 3, 5, … .
2. Дано целое число N (> 0). Сформировать и вывести целочисленный массив размера N, содержащий степени двойки от первой до N-ый: 2, 4, 8, 16, … .
3. Дано целое число N (> 1), а также первый член A и разность D арифметической прогрессии. Сформировать и вывести массив размера N, содержащий N первых членов данной прогрессии: A, A + D, A + 2·D, A + 3·D, … .
4. Дано целое число N (> 1), а также первый член A и знаменатель D геометрической прогрессии. Сформировать и вывести массив размера N, содержащий N первых членов данной прогрессии: A, A·D, A·D2, A·D3, … .
5. Дано целое число N (> 2). Сформировать и вывести целочисленный массив размера N, содержащий N первых элементов последовательности чисел Фибоначчи FK: F1 = 1, F2 = 1, FK = FK–2 + FK–1, K = 3, 4, … .
6. Даны целые числа N (> 2), A и B. Сформировать и вывести целочисленный массив размера N, первый элемент которого равен A, второй равен B, а каждый последующий элемент равен сумме всех предыдущих.
7. Дан массив размера N. Вывести его элементы в обратном порядке.
8. Дан целочисленный массив размера N. Вывести все содержащиеся в данном массиве нечетные числа в порядке возрастания их индексов, а также их количество K.
9. Дан целочисленный массив размера N. Вывести все содержащиеся в данном массиве четные числа в порядке убывания их индексов, а также их количество K.

10. Дан целочисленный массив размера N. Вывести вначале все содержащиеся в данном массиве четные числа в порядке возрастания их индексов, а затем — все нечетные числа в порядке убывания их индексов.
11. Дан массив A размера N и целое число K ($1 \le K \le N$). Вывести элементы массива с порядковыми номерами, кратными K: AK, A2·K, A3·K, … . Условный оператор не использовать.
12. Дан массив A размера N (N — четное число). Вывести его элементы с четными номерами в порядке возрастания номеров: A2, A4, A6, …, AN. Условный оператор не использовать.
13. Дан массив A размера N (N — нечетное число). Вывести его элементы с нечетными номерами в порядке убывания номеров: AN, AN–2, AN–4, …, A1. Условный оператор не использовать.
14. Дан массив A размера N. Вывести вначале его элементы с четными номерами (в порядке возрастания номеров), а затем — элементы с нечетными номерами (также в порядке возрастания номеров): A2, A4, A6, …, A1, A3, A5, … . Условный оператор не использовать.
15. Дан массив A размера N. Вывести вначале его элементы с нечетными номерами в порядке возрастания номеров, а затем — элементы с четными номерами в порядке убывания номеров. A1, A3, A5, …, A6, A4, A2. Условный оператор не использовать.
16. Дан массив A размера N. Вывести его элементы в следующем порядке: A1, AN, A2, AN–1, A3, AN–2, … .
17. Дан массив A размера N. Вывести его элементы в следующем порядке: A1, A2, AN, AN–1, A3, A4, AN–2, AN–3, … .
18. Дан массив A ненулевых целых чисел размера 10. Вывести значение первого из тех его элементов AK, которые удовлетворяют неравенству AK < A10. Если таких элементов нет, то вывести 0.
19. Дан целочисленный массив A размера 10. Вывести порядковый номер последнего из тех его элементов AK, которые удовлетворяют двойному неравенству A1 < AK < A10. Если таких элементов нет, то вывести 0.
20. Дан массив размера N и целые числа K и L ($1 \le K \le L \le N$). Найти сумму элементов массива с номерами от K до L включительно.
21. Дан массив размера N и целые числа K и L ($1 \le K \le L \le N$). Найти среднее арифметическое элементов массива с номерами от K до L включительно.
22. Дан массив размера N и целые числа K и L ($1 < K \le L \le N$). Найти сумму всех элементов массива, кроме элементов с номерами от K до L включительно.
23. Дан массив размера N и целые числа K и L ($1 < K \le L \le N$). Найти среднее арифметическое всех элементов массива, кроме элементов с номерами от K до L включительно.
24. Дан целочисленный массив размера N, не содержащий одинаковых чисел. Проверить, образуют ли его элементы арифметическую прогрессию. Если образуют, то вывести разность прогрессии, если нет — вывести 0.
25. Дан массив ненулевых целых чисел размера N. Проверить, образуют ли его элементы геометрическую прогрессию. Если образуют, то вывести знаменатель прогрессии, если нет — вывести 0.

Практическая работа №32 «Использование элементов управления отображение данных в таблице и методов на C#»

Составить программу с использованием элемента управления DataGridView и операторов языка программирования C#. Оформить форму и все ее элементы.

1. Дана матрица размера M × N и целое число K (1 ≤ K ≤ M). Вывести элементы K-й строки данной матрицы.
2. Дана матрица размера M × N и целое число K (1 ≤ K ≤ N). Вывести элементы K-го столбца данной матрицы.
3. Дана матрица размера M × N. Вывести ее элементы, расположенные в строках с четными номерами (2, 4, …). Вывод элементов производить по строкам, условный оператор не использовать.
4. Дана матрица размера M × N. Вывести ее элементы, расположенные в столбцах с нечетными номерами (1, 3, …). Вывод элементов производить по столбцам, условный оператор не использовать.
5. Дана матрица размера M × N. Вывести ее элементы в следующем порядке: первая строка слева направо, вторая строка справа налево, третья строка слева направо, четвертая строка справа налево и т. д.
6. Дана матрица размера M × N. Вывести ее элементы в следующем порядке: первый столбец сверху вниз, второй столбец снизу вверх, третий столбец сверху вниз, четвертый столбец снизу вверх и т. д.
7. Дана квадратная матрица A порядка M. Начиная с элемента A1,1, вывести ее элементы следующим образом («уголками»): все элементы первой строки; элементы последнего столбца, кроме первого (уже выведенного) элемента; оставшиеся элементы второй строки; оставшиеся элементы предпоследнего столбца и т. д.; последним выводится элемент AM,1.
8. Дана квадратная матрица A порядка M. Начиная с элемента A1,1, вывести ее элементы следующим образом («уголками»): все элементы первого столбца; элементы последней строки, кроме первого (уже выведенного) элемента; оставшиеся элементы второго столбца; оставшиеся элементы предпоследней строки и т. д.; последним выводится элемент A1,M.
9. Дана квадратная матрица A порядка M (M — нечетное число). Начиная с элемента A1,1 и перемещаясь по часовой стрелке, вывести все ее элементы по спирали: первая строка, последний столбец, последняя строка в обратном порядке, первый столбец в обратном порядке, оставшиеся элементы второй строки и т. д.; последним выводится центральный элемент матрицы.
10. Дана квадратная матрица A порядка M (M — нечетное число). Начиная с элемента A1,1 и перемещаясь против часовой стрелки, вывести все ее элементы по спирали: первый столбец, последняя строка, последний столбец в обратном порядке, первая строка в обратном порядке, оставшиеся элементы второго столбца и т. д.; последним выводится центральный элемент матрицы.
11. Дана матрица размера M × N и целое число K (1 ≤ K ≤ M). Найти сумму и произведение элементов K-й строки данной матрицы.
12. Дана матрица размера M × N и целое число K (1 ≤ K ≤ N). Найти сумму и произведение элементов K-го столбца данной матрицы.
13. Дана матрица размера M × N. Для каждой строки матрицы найти сумму ее элементов.
14. Дана матрица размера M × N. Для каждого столбца матрицы найти произведение его элементов.
15. Дана матрица размера M × N. Для каждой строки матрицы с нечетным номером (1, 3, …) найти среднее арифметическое ее элементов. Условный оператор не использовать.
16. Дана матрица размера M × N. Для каждого столбца матрицы с четным номером (2, 4, …) найти сумму его элементов. Условный оператор не использовать.
17. Дана матрица размера M × N. В каждой строке матрицы найти минимальный элемент.

18. Дана матрица размера M × N. В каждом столбце матрицы найти максимальный элемент.
19. Дана матрица размера M × N. Найти номер ее строки с наибольшей суммой элементов и вывести данный номер, а также значение наибольшей суммы.
20. Дана матрица размера M × N. Найти номер ее столбца с наименьшим произведением элементов и вывести данный номер, а также значение наименьшего произведения.
21. Дана матрица размера M × N. Найти максимальный среди минимальных элементов ее строк.
22. Дана матрица размера M × N. Найти минимальный среди максимальных элементов ее столбцов.
23. Дана матрица размера M × N. В каждой ее строке найти количество элементов, меньших среднего арифметического всех элементов этой строки.
24. Дана матрица размера M × N. В каждом ее столбце найти количество элементов, больших среднего арифметического всех элементов этого столбца.
25. Дана матрица размера M × N. Найти номера строки и столбца для элемента матрицы, наиболее близкого к среднему значению всех ее элементов.

Практическая работа №16 «Создание классов в C#»

Составить программу элементов управления и операторов языка программирования C#. Оформить форму и все ее элементы. Оформить форму и все ее элементы, используя цветовую гамму.

1. Описать структуру с именем GROUP, содержащую поля: Name – фамилия и инициалы, DAT – дата рождения (год, месяц, число), SES – успеваемость (массив из трех элементов).

Написать программу, выполняющую:

ввод с клавиатуры данных в массив GR5,состоящий из 10 структур типа GROUP;

– вывод на экран записей, упорядоченных по возрастанию поля SES; – вывод списка студентов, возраст которых на 01.12.2010 года не превышает 20 лет;

– если таких студентов нет – выдать сообщение.

2. Описать структуру с именем STUDENT, содержащую поля: Name – фамилия и инициалы, Kurs − курс, SES – успеваемость (массив из пяти элементов).

Написать программу, выполняющую:

ввод с клавиатуры данных в массив STUD, состоящий из 10 структур типа STUDENT, записи должны быть упорядочены по алфавиту;

– вывод на экран записей, упорядоченного списка студентов, средний бал которых превышает общий средний бал;

– если таких студентов нет – выдать сообщение.

3. Описать структуру с именем STUD, содержащую поля: Name – фамилия и инициалы, GROUP – название группы (факультет, курс, номер группы), SES – успеваемость (массив из четырёх элементов).

Написать программу, выполняющую:

– ввод с клавиатуры данных в массив STUD1,состоящий из 10 структур

типа STUD, записи должны быть упорядочены по алфавиту;

– вывод на экран данных о студентах, включенных в массив, средний бал которых превышает 4,2. Список упорядочить по возрастанию среднего бала. Сохранить информацию о положении студента в исходном списке;

– если таких студентов нет – выдать сообщение.

4. Описать структуру с именем NOTE, содержащую поля: Name – фамилия и инициалы, TELE – номер телефона, DATE – дата рождения (год, месяц, число).

Написать программу, выполняющую:

ввод с клавиатуры данных в массив BLOCKNOTE, состоящий из 10 структур типа NOTE, записи должны быть упорядочены по возрастанию даты рождения;
– вывод на экран сведений о человеке, номер телефона которого введен с клавиатуры;
– если такого человека нет – выдать сообщение.
5. Описать структуру с именем NOTE1, содержащую поля: Name – фамилия и инициалы, TELE – номер телефона, DATE – дата рождения (год, месяц, число.
Написать программу, выполняющую:
– ввод с клавиатуры данных в массив BLOCK, состоящий из 9 элементов типа NOTE1, записи должны быть упорядочены по инициалам;
– вывод на экран информации о людях, чьи дни рождения приходятся на месяц, значение которого введено с клавиатуры; если такого человека нет – выдать сообщение.
6. Описать структуру с именем NOTE2, содержащую поля: Name – фамилия и инициалы, TELE – номер телефона, DATE – дата рождения (год, месяц, число).
Написать программу, выполняющую:
– ввод с клавиатуры данных в массив BLOCK2, состоящий из 7 элементов типа NOTE1, записи должны быть упорядочены по первым трем цифрам номера телефона;
– вывод на экран информации о человеке, чья фамилия введена с клавиатуры;
– если такого нет – выдать сообщение.
7. Описать структуру с именем PERSON, содержащую поля: Name – фамилия и инициалы, FAC – факультет, GROUP – группа, DATE – дата поступления в ВУЗ (год, месяц, число).
Написать программу, выполняющую:
– ввод с клавиатуры данных в массив VUZ, состоящий из 10 элементов
– типа PERSON, записи должны быть упорядочены по дате поступления в ВУЗ;
– вывод на экран информации о студентах, упорядоченной по факультетам, группам, дате поступления. В каждой группе фамилии должны быть расположены в алфавитном порядке.
8. Описать структуру с именем ZNAK, содержащую поля: Name – фамилия и имя, ZOD – знак зодиака, DATE – дата рождения (массив из трех чисел: год, месяц, число).
Написать программу, выполняющую:
– ввод с клавиатуры данных в массив MASS, состоящий из 10 элементов типа ZNAK, записи должны быть упорядочены по дате дня рождения;
– вывод на экран информации о людях, родившихся под знаком зодиака, наименование которых вводится с клавиатуры;
– если такого нет – выдать сообщение.
9. Структура содержит информацию о дате и времени некоторого события: Year – год, Day – день, Hour – часы, Minute – минуты, Second – секунды.
Написать программу, выполняющую:
– определение размера структурированного объекта в батах.
– записывает предложенную структуру в виде битовой структуры и определяет размеры.
Сравните результаты, сделайте вывод.
10. Для хранения данных о цветных дисплеях описать структуру вида: display – наименование модели, price – цена, x_size – размер по горизонтали, y_size – размер по вертикали, optr – оптическое разрешение.
Написать функцию, создающую файл с данными о дисплеях (данные вводить с клавиатуры) из не менее восьми записей, осуществляющую его сортировку по заданному параметру (обязательный параметр – признак, задающий критерий сортировки). Все

необходимые данные для функции должны передаваться ей в качестве параметров. Использование глобальных параметров не допускается.

11. Описать структуру с именем STUDENT, содержащую поля: фамилия и инициалы студента; номер группы; успеваемость (массив из четырех оценок на экзаменах в 5-бальной системе).

Написать функции:

– создания массива 7 записей (структур) данных о студентах (ввод данных с клавиатуры);

– вычисления среднего бала каждого студента;

– расположения записей по убыванию среднего бала;

– вывода сведений о студентах, имеющих оценки только 4 и 5;

– удаления из списка студента с минимальным средним балом.

Все необходимые данные для функций должны передаваться в качестве их параметров. Использование глобальных параметров не допускается. Создать проект, который демонстрирует работу всех функций.

12. Описать структуру с именем TOVAR, содержащую поля: название товара; количество единиц товара; стоимость товара; дата поступления товара в виде структуры (год, месяц, день).

Написать функции:

– создания массива SPISOK не более чем из 10 записей (структур) данных о товарах (ввод данных с клавиатуры);

– вычисления средней стоимости товара;

– расположения записей по возрастанию стоимости товаров;

– вывода сведений о товарах, поступивших более 10 месяцев назад.

Все необходимые данные для функций должны передаваться в качестве их параметров. Использование глобальных параметров не допускается. Создать проект, который демонстрирует работу всех функций.

13. Описать структуру с именем MARSHRUT, содержащую поля: номер маршрута; начальный пункт маршрута; конечный пункт маршрута; длина маршрута.

Написать функции:

– создания массива не более чем из 10 записей (структур) сведений о маршрутах (ввод данных с клавиатуры);

– определения маршрута с максимальной длиной;

– расположения записей по возрастанию номеров маршрутов;

– вывода сведений о маршрутах, которые начинаются или заканчиваются в пункте, название которого вводится с клавиатуры.

Все необходимые данные для функций должны передаваться в качестве их параметров. Использование глобальных параметров не допускается. Создать проект, который демонстрирует работу всех функций.

14. Описать структуру с именем ABON, содержащую поля: фамилия и инициалы абонента, номер телефона, дата подключения телефона в виде структуры (год, месяц, день), начисленная сумма оплаты, сумма на счету абонента.

Написать функции:

– создания массива не более чем из 12 записей (структур) данных об абонентах (ввод данных с клавиатуры);

– расположения записей по алфавиту (с учетом инициалов для абонентов с одинаковыми фамилиями;

– добавить 20 гр. на счета абонентов, которых подключили более 10 лет назад;
– вывода сведений об абонентах, у которых сумма на счету отрицательная после вычета начислений;
– вывода сведений об абоненте, номер телефона которого вводится с клавиатуры.
Все необходимые данные для функций должны передаваться в качестве их параметров. Использование глобальных параметров не допускается. Создать проект, который демонстрирует работу всех функций.
15. Описать структуру, содержащую поля: наименование, цена, дата производства, срок годности, количество, производитель.
Написать функции, выполняющие следующие задачи:
вывода сведений о товарах, срок годности которых менее 20-ти дней.
– определить количество просроченных товаров.
16. Описать структуру, содержащую поля: марка, автомобиля, производитель, тип, год выпуска, дата прохождения техосмотра, дата регистрации.
Написать функции, выполняющие следующие задачи:
– вывода сведений о машинах, прошедших техосмотр менее года назад.
– вывода сведений о машинах, произведенных до 2000-го года и зарегистрированных менее года назад.
17. Описать структуру, содержащую поля: фамилия, амплуа, возраст, количество игр, количество голов.
Написать функции, выполняющие следующие задачи:
– определить лучшего форварда, и вывести сведения о футболистах, сыгравших менее 5-ти игр.
– определить средний возраст футболистов и вывести сведения о футболистах, возраст которых больше 25 лет.
18. Описать структуру, содержащую поля: фамилия, группа, физика, информатика, история.
Написать функции, выполняющие следующие задачи:
– определить средний бал оценок по всем предметам, и вывести сведе-ния о студентах, средний балл которых больше 4.
– определить средний бал оценок по физике, количество студентов с оценкой 5 по информатике и вывести сведения о них.
19. Описать структуру, содержащую поля: продавец, наименование, количество, цена, год_выпуска, дата_продажи.
Написать функции, выполняющие следующие задачи:
– определить количество товаров, которые проданы менее года назад и вывести сведения о них.
– определить количество всех товаров, количество которых больше 5 и вывести сведения об этих товарах.
– определить общую стоимость всех товаров, выпущенных в текущем году и вывести сведения об этих товарах.
20. Описать структуру, содержащую поля: автор, количество, страниц, тираж, год, издания.
– написать функции, выполняющие следующие задачи:
– вывести данные о книгах, в которых количество страниц больше 150.
– вывести данные о книгах, тираж которых не превышает 10000 экземпляров.

21. Описать структуру, содержащую поля: наименование, производитель, год_выпуска, количество, цена.
Написать функции, выполняющие следующие задачи:
– вывести сведения о товарах с ценой выше средней.
– определить общую стоимость всех товаров, выпущенных в текущем году и вывести сведения об этих товарах.
22. Описать структуру, содержащую поля: фамилия, группа, год рождения, оценка по физике, оценка по математике, оценка по информатике.
Написать функции, выполняющие следующие задачи:
– определить количество студентов старше 19-ти лет, и напечатать все сведения о них.
– напечатать фамилии студентов, которые сдали математику на «95», и определить их количество.
23. Описать структуру, содержащую поля: название, частота, объем оперативной памяти, наличие dvd rom, стоимость.
Написать функции, выполняющие следующие задачи:
– определить количество компьютеров с объемом оперативной памяти больше 10 гбайт и напечатать все сведения о них.
– вычислить среднюю стоимость всех компьютеров и напечатать наименования компьютеров и их среднюю стоимость.
– определить компьютеры, которые имеют dvd rom, и напечатать все сведения о них.
24. Описать структуру, содержащую поля: фамилия, имя, отчество, пол, должность, дата рождения.
Написать функции, выполняющие следующие задачи:
– вывести данные об инженерах, пенсионного возраста (мужчинам больше 65-ти лет, женщинам 60).
– вывести сведения о сотрудниках, у которых зарплата выше средней и возраст менее 30-ти лет.
– вывести сведения о сотрудниках, которые родились в мае.
25. Описать структуру, содержащую поля: № поезда, время прибытия, время отбытия, направление, расстояние.
Написать функции, выполняющие следующие задачи:
– вывести среднюю скорость каждого поезда.
– вывести все сведения о поездах, время пребывания в пути которых превышает 7 часов.

ПРАКТИЧЕСКИЕ РАБОТЫ ПО VISUAL STUDIO

ИЗУЧЕНИЕ СРЕДЫ РАЗРАБОТКИ VISUAL STUDIO

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

Интегрированная среда разработчика Visual Studio

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

При запуске Visual Studio появляется начальная страница со спи- ском последних проектов, а также командами *Создать проект* и *От- крыть проект*. Нажмите ссылку *Создать проект* или выберите в меню *Файл* команду *Создать проект*, на экране появится диалог для создания нового проекта (рис. 1.1).

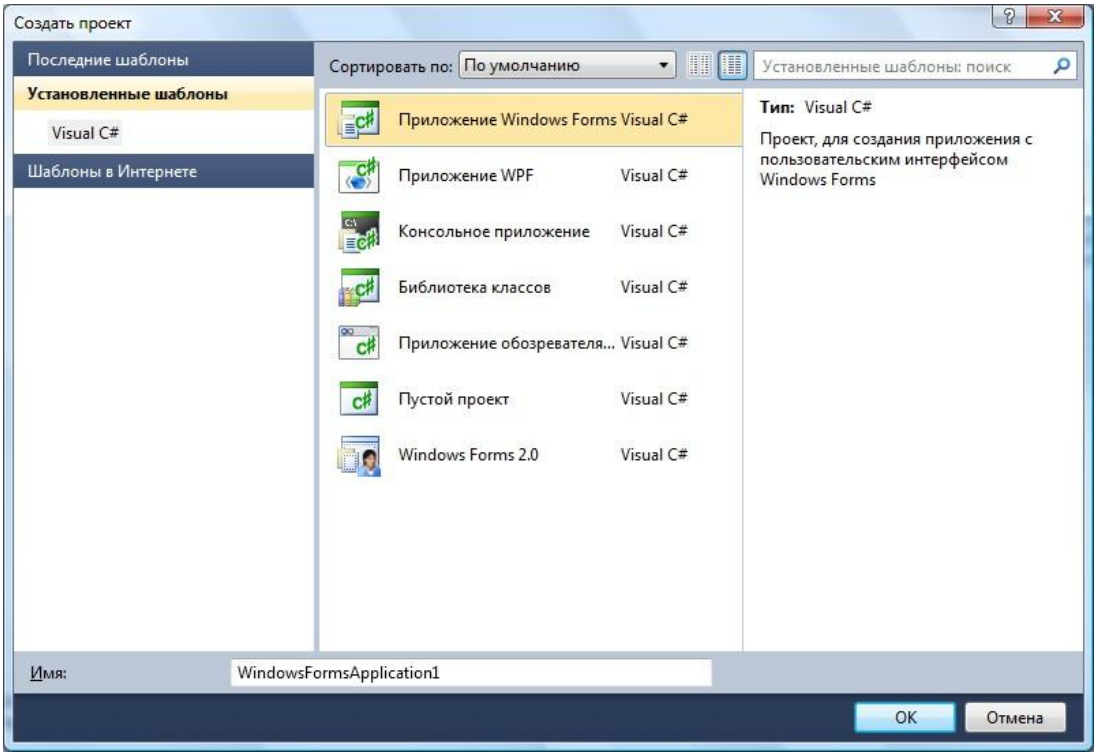


Рис. 1.1. Диалог создания нового проекта

Слева в списке шаблонов приведены языки программирования, ко- торые 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 исполь- зоваться два типа проектов:

Приложение Windows Forms – данный тип проекта позволяет создать полноценное приложение с окнами и элементами управления (кнопками, полями ввода и пр.) Такой вид приложения наиболее привы- чен большинству пользователей.

Консольное приложение – в этом типе проекта окно представ- ляет собой текстовую консоль, в которую приложение может выводить тексты или ожидать ввода информации пользователя. Консольные при- ложения 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, подоб- ную представленной на рис. 1.2.

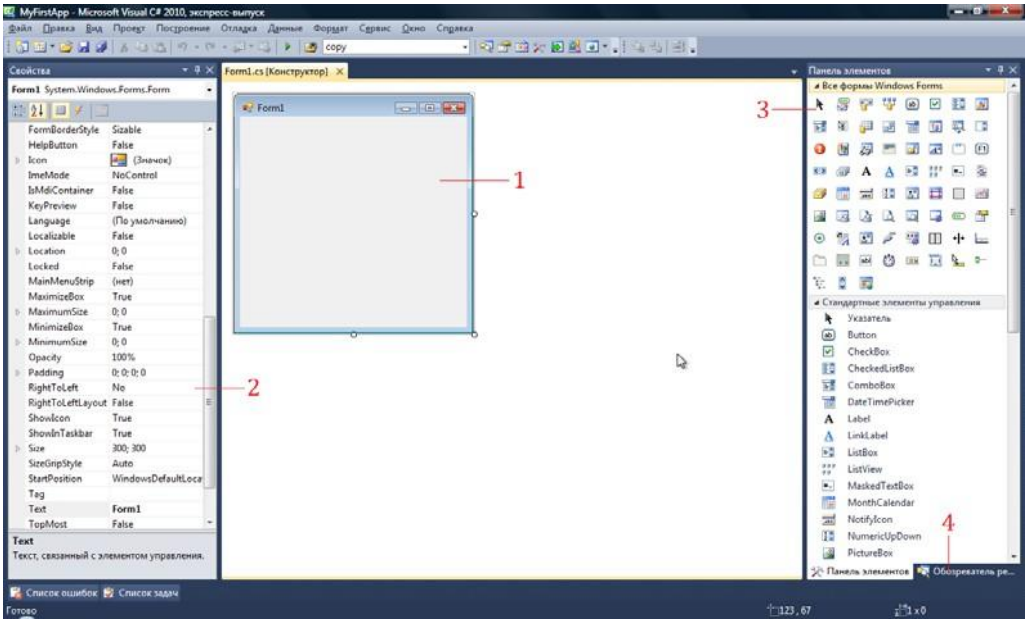


Рис. 1.2. Главное окно Visual Studio

В главном окне Visual Studio присутствует несколько основных эле- ментов, 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 элемен- ты управления будут иметь тот же вид, что и на этапе проектирования.

Второй по важности объект – это окно свойств (2), в котором при- ведены 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 активи- ровать 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, что- бы 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 нор- мального 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 дос- тупа к элементу управления (в скрытой части класса формы).

Программа на языке C# составляется как описание алгоритмов, которые необходимо выполнить, если возникает определенное собы- тие, связанное с формой (например, щелчок «мыши» на кнопке – собы- тие 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# тре- буемый алгоритм.

Настройка формы

Настройка формы начинается с настройки размера формы. С по- мощью мыши, «захватывая» одну из кромок формы или выделенную строку заголовка, отрегулируйте нужные размеры формы.

Для настройки будущего окна приложения задаются свойства фор- мы. 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 ок- не 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.

Размещение элементов управления на форме

Для размещения различных элементов управления на форме ис- пользуется 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 управле- ния из группы *Стандартные элементы управления*.

Щелкните на элементе управления, который хотите добавить, а за- тем щелкните в нужном месте формы – элемент появится на форме. Эле- мент 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 раз- мещения элемента управления на форме, его можно выделить щелчком мыши и при этом получить доступ к его свойствам в окне свойств.

Размещение строки ввода

Если необходимо ввести из формы в программу или вывести на форму информацию, которая вмещается в одну строку, используют ок- но однострочного редактора текста, представляемого элементом управ- ления 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 использо- вать 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.

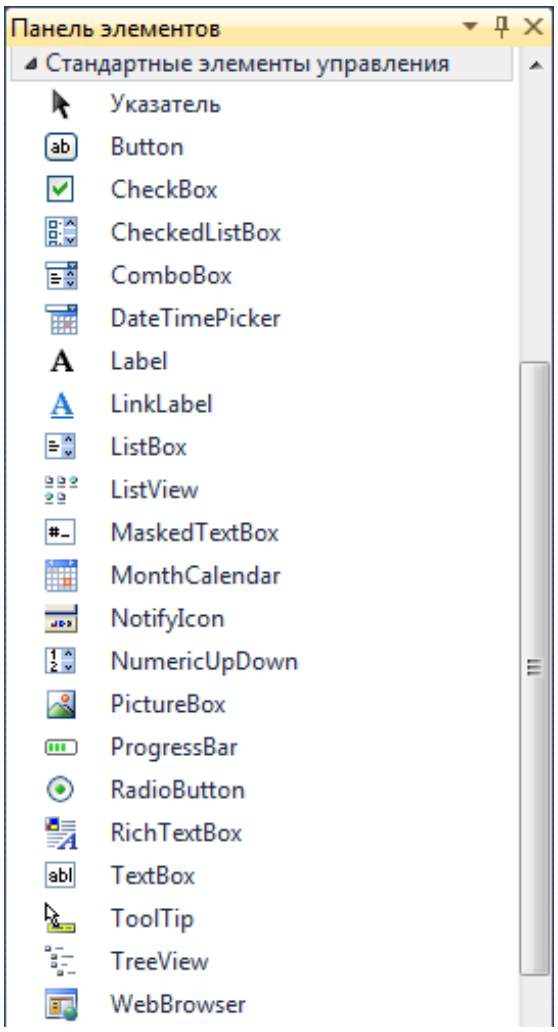


Рис. 1.3. Панель элементов

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

Размещение надписей

На форме могут размещаться пояснительные надписи. Для нане- сения 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, вве- дите строку, например «*Введите свое имя:*», а также выберите размер символов (свойство Font).

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

Написание программы обработки события

С каждым элементом управления на форме и с самой формой могут происходить события во время работы программы. Например, с кноп- кой может произойти событие – нажатие кнопки, а с окном, которое проектируется с помощью формы, может произойти ряд событий: соз- дание окна, изменение размера окна, щелчок мыши на окне и т. п. Эти события могут обрабатываться в программе. Для обработки таких собы- тий 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 обработ- чик события нажатия.

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 по- ложение 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 до- бавлять свой код между скобками { }. Например, наберите:

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

Написание программы обработки события загрузки формы

Второй способ создания обработчика события заключается в выбо- ре соответствующего события для выделенного элемента на форме. При этом используется окно свойств и его закладка  . Рассмотрим этот спо- соб. Выделите форму щелчком по ней, чтобы вокруг нее появилась рам- ка 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 собы- тий, однако некоторые из них присущи большинству элементов управ- ления. Наиболее часто применяемые события описаны ниже:
Activated: форма получает это событие при активации.
Load: возникает при загрузке формы. В обработчике данного собы- тия следует задавать действия, которые должны происходить в мо- мент создания формы, например установка начальных значений.
KeyPress: возникает при нажатии кнопки на клавиатуре. Параметр e.KeyChar имеет тип char и содержит код нажатой клавиши (кла- виша 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. Обработ- чик этого события получает информацию о нажатой клавише и со- стоянии 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 ин- формацию о клавишах-модификаторах Shift и др. можно узнать из параметра e.Modifiers.
KeyUp: является парным событием для KeyDown и возникает при от- пускании ранее нажатой клавиши.
Click: возникает при нажатии кнопки мыши в области элемента управления.
DoubleClick: возникает при двойном нажатии кнопки мыши в об- ласти элемента управления.

Запуск и работа с программой

Запустить программу можно, выбрав в меню *Отладка* команду *На- чать отладку*. При этом происходит трансляция и, если нет ошибок, компоновка программы и создание единого загружаемого файла с рас- ширением .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, одна- ко, 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 за- претить запуск при наличии ошибок (рис. 1.4 и 1.5).

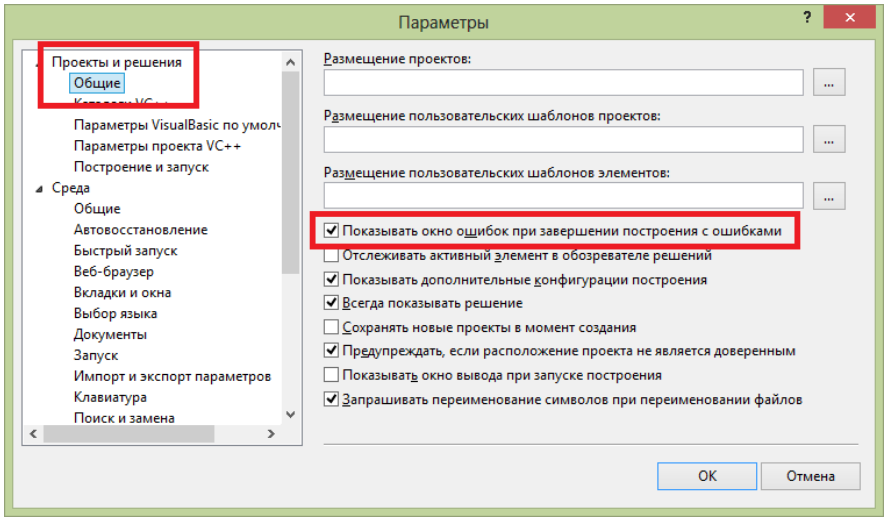


Рис. 1.4. Опция показа сообщений об ошибках

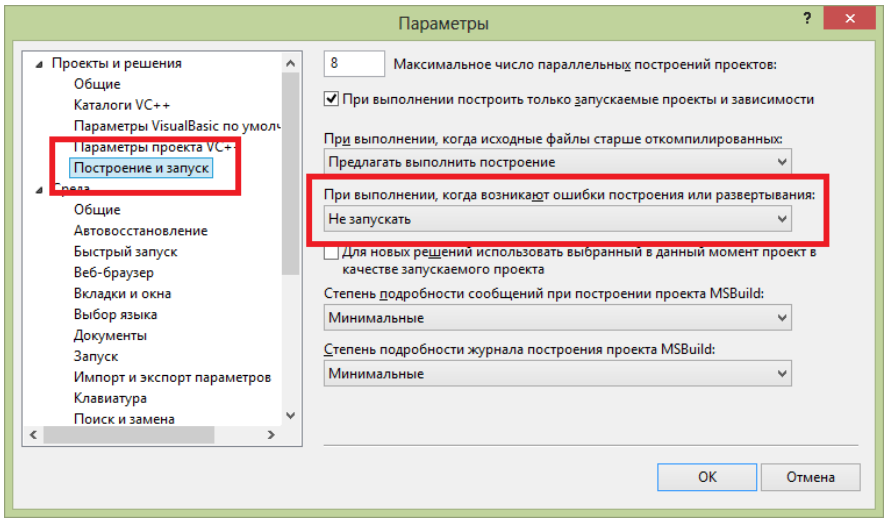


Рис. 1.5. Опция отключения запуска предыдущей версии при ошибках построения

Для завершения работы программы и возвращения в режим проек- тирования формы не забудьте закрыть окно программы!

Динамическое изменение свойств

Свойства элементов на окне могут быть изменены динамически во время выполнения программы. Например, можно изменить текст надпи- си или цвет формы. Изменение свойств происходит внутри обработчика события (например, обработчика события нажатия на кнопку). Для это- го 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 ок- не свойств в свойстве (Name) для текущего элемента. Допускается ис-

пользование 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;
```

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 элемен- тов управления Form, Label, TextBox, Button. С помощью окна свойств ус- тановите первоначальный цвет формы, шрифт выводимых символов.

Индивидуальные задания

Разместите на форме четыре кнопки (Button). Сделайте на кнопках следующие надписи: «красный», «зеленый», «синий», «жел- тый». Создайте четыре обработчика события нажатия на данные кнопки, которые будут менять цвет формы в соответствии с текстом на кнопках.

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

Разместите на форме ряд кнопок (Button) напротив каждой по- ле 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.

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.

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

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.

Разместите на форме ряд кнопок (Button), напротив каждой по- ле ввода (TextBox). Создайте обработчики события нажатия на данные кнопки, которые будут менять заголовок окна. Текст в заголовке берет- ся из поля ввода напротив нажимаемой кнопки.

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

Разместите на форме три кнопки (Button) и одно поле ввода (TextBox). Сделайте на кнопках следующие надписи: «скрыть», «пока- зать», «очистить». Создайте обработчики события нажатия на данные кнопки, которые будут скрывать или показывать поле ввода. При нажа- тии на кнопку «очистить» текст из поля ввода должен быть удален.

Разместите на форме две кнопки (Button) и одно поле ввода (TextBox). Сделайте на кнопках следующие надписи: «заполнить», «очи- стить». Создайте обработчики события

нажатия на данные кнопки, кото- рые будут очищать или заполнять поле ввода знаками «******». Создайте обработчик события создания формы (Load), который будет устанавливать цвет формы и менять текст в поле ввода на строку «+++++».

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

Разработайте игру, которая заключается в следующем. На фор- ме размещены четыре кнопки (Button) и четыре метки (Label). При на- жатии на кнопку часть надписей становится невидимой, а часть, наобо- рот, становится видимой. Цель игры – скрыть все надписи.

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

Разместите на форме ряд кнопок (Button). Создайте обработчи- ки события нажатия на данные кнопки, которые будут делать неактив- ными следующую кнопку. Создайте обработчик события нажатия кноп- ки мыши на форме (Click), который будет устанавливать все кнопки в активный режим.

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

Разместите на форме ряд полей ввода (TextBox). Создайте об- работчики события нажатия кнопкой мыши на данные поля ввода, кото- рые будут выводить в текущее поле ввода его номер. Создайте обработ- чик события изменения размера формы (Resize), который будет очищать все поля ввода.

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

Разместите на форме поле ввода (TextBox) и две кнопки (Button) с надписями: «блокировать», «разблокировать». Создайте об- работчики 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.

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

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

КЛАССЫ И ОБЪЕКТЫ

Цель практической работы: изучить основные понятия, относя- щиеся к классам и объектам, освоить динамическое создание объектов в программном коде.

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 описа- ния переменных, свойств и методов, которые относятся к объекту. Примером класса в реальной жизни является *понятие*

«автомобиль»: как правило, автомобиль содержит некоторое количество колес, две- рей, имеет какой-то цвет, но эти конкретные детали в классе не опи- сываются.
Объект – это экземпляр класса. Свойства объекта содержат кон- кретные данные, характерные для данного экземпляра. В реальной жиз- ни примером объекта будет конкретный экземпляр автомобиля с 4 коле- сами, 5 дверками и синего цвета.

Динамическое создание объектов

Чаще всего для размещения на форме кнопки, поля ввода или дру- гих управляющих элементов используется дизайнер среды Visual Studio: нужный элемент выделяется в панели элементов и размещается на фор- ме. Однако иногда создавать элементы нужно уже в процессе выполне- ния программы. Поскольку каждый элемент управления представляет собой отдельный класс, его помещение на форму программным спосо- бом включает несколько шагов:
Создание экземпляра класса.
Привязка его к форме.
Настройка местоположения, размеров, текста и т. п.
Например, чтобы создать кнопку, нужно выполнить следующий код (его следует разместить в обработчике сообщения Load или в каком- либо другом методе):

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

Здесь объявляется переменная b, относящаяся к классу Button, как и в предыдущих лабораторных работах. Однако дальше идет нечто но- вое: 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 разме- щен выполняемый в данный момент метод. Поскольку все методы в ла- бораторных работах размещаются в классе формы, то и this относится к этому конкретному экземпляру формы.
Вместо формы кнопку можно поместить на другой контейнер. На- пример, если на форме есть элемент управления Panel, то можно помес- тить 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 эле- мент управления или вносятся какие-то изменения в свойства элементов управления, Visual Studio генерирует специальный служебный код, кото- рый 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 вет- ку формы Form1, разверните ее и сделайте двойной щелчок по ветке Form1.Designer.cs. Откроется файл с текстом программы на языке C#, ко- торую среда создала автоматически. Менять этот код вручную крайне не рекомендуется! Однако можно его изучить, чтобы понять принципы соз- дания элементов управления в ходе выполнения программы.

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:

Переменные, описанные внутри метода, не будут видны за пре- делами этого метода. Например:

```
void MethodA()
{
// Описываем переменную delta int delta = 7;
}
void MethodB()
{
// Ошибка: переменная delta в этом методе неизвестна! int gamma = delta + 1;
}
```

Переменные, описанные внутри блока или составного операто- ра, 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());
}
```

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());
}
}
```

Операции is и as

Часто бывает удобно переменные разных классов записать в один спи- сок, чтобы было легче его обрабатывать. Чтобы проверить, к какому классу принадлежит какой-либо объект, можно использовать оператор is: он воз- вращает 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 рас- ширенным 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 опе- ратор as выбросит исключение и остановит выполнение программы. Поэтому более безопасный подход состоит в комбинированном приме- нении 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 управле- ния!) 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 до- полнительные 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 пере- менных, 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 мо- мент нажатия, в виде объекта класса Point;

Хи Y– те же координаты в виде отдельных переменных.

Индивидуальные задания

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

Разработать программу, динамически порождающую на окне кнопки. Левый верхний угол кнопки определяется местоположением курсора при щелчке. Вывести надпись на кнопке с координатами ее ле- вого верхнего угла.

Разработать программу, динамически порождающую на окне кноп- ки и поля ввода. Левый верхний угол элемента управления определяется ме- стоположением курсора при щелчке. Кнопка порождается, если курсор на- ходится в левой половине окна, в ином случае порождается поле ввода.

На форме размещен элемент управления Panel. Написать про- грамму, которая при щелчке мыши на элементе управления Panel до- бавляет в него кнопки Button, а при щелчке на форме в нее добавляются поля ввода TextBox.

На форме размещены 3 панели (элемент управления Panel). На- писать программу, которая при щелчке мыши на первой панели добав- ляет во вторую панель кнопки Button, при щелчке на второй панели до- бавляет в третью панель поля ввода TextBox, а при щелчке на третьей панели добавляет на первую панель метки Label.

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.

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

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

Создать программу, добавляющую различные элементы управ- ления на форму и на панель Panel. Тип элементов управления выбира- ется случайным образом. Предусмотреть возможность вывода инфор- мации о количестве элементов по типам и информацию о расположении элементов.

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

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

Написать программу, динамически порождающую на окне по- очередно кнопки или поля ввода. Левый верхний угол элемента управ- ления определяется местоположением курсора при щелчке. При нажа- тии правой кнопки мыши на форме с нее удаляются все порожденные элементы.

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

Разработать программу с двумя кнопками на форме. При нажа- тии на первую на форму добавляется одна кнопка или поле ввода. При нажатии на вторую кнопку каждое поле увеличивается по вертика- ли в два раза.

Написать программу с кнопкой и тремя полями ввода. При на- жатии 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 содержат- ся координаты левого верхнего угла будущего элемента управления.

Разработать программу, добавляющую на форму метки с тек- стом. Местоположение и размеры меток определяются в программе ди- намически 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.

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 нажа- тии на вторую кнопку каждое поле уменьшается по вертикали в два раза.

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.

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.

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

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

АНИМАЦИЯ

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

Работа с таймером

Класс для работы с таймером Timer формирует в приложении по- вторяющиеся события. События повторяются с периодичностью, ука- занной в миллисекундах в свойстве Interval. Установка свойства Enabled в значение true запускает таймер. Каждый тик таймера порож- дает событие Tick, обработчик которого обычно и создают в приложе- нии. В этом обработчике могут изменяться какие-либо величины и вы- зываться принудительная перерисовка окна. Для создания анимации весь код, рисующий что-либо на форме, должен находиться в обработ- чике события Paint.

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();
}
```

Движение по траектории

Движение по траектории реализуется аналогично вышерассмот- ренному примеру. Для реализации движения по прямой нужно увеличи- вать переменные, являющиеся узловыми точками, на определенные константы: в приведенном выше примере это переменные x2 и y2. Для задания более сложной траектории можно использовать различные па- раметрические кривые.

В случае движения на плоскости обычно изменению подвергает- ся один параметр. Рассмотрим пример реализации движения окруж- ности по *декартову листу*. Декартов лист – это плоская кривая третьего порядка, удовлетворяющая уравнению в прямоугольной сис- теме $x^3 + y^3 = 3 \cdot a \cdot x \cdot y$. 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 $t = \operatorname{tg} \varphi$.

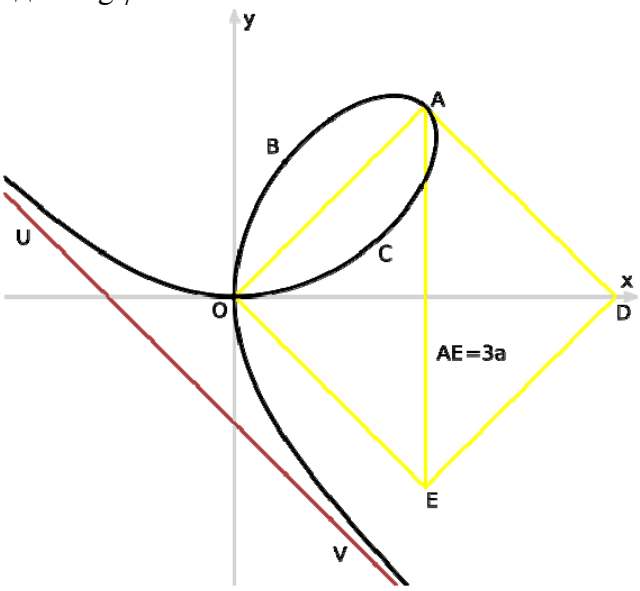


Рис. 11.1. Декартов лист

Описание ряда интересных кривых для создания траектории дви- жения 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

Создайте программу, показывающую пульсирующее сердце.
Создайте приложение, отображающее вращающийся винт са- молета.
Разработайте программу анимации двигающегося человечка.
Создайте программу, показывающую движение окружности по синусоиде.
Создайте приложение, отображающее движение окружности по спирали.
Разработайте программу анимации падения снежинки.
Создайте программу, показывающую скачущий мячик.
Создайте приложение, отображающее движение окружности вдоль границы окна. Учтите возможность изменения размеров окна.
Разработайте программу анимации летающего бумеранга.
Создайте программу, показывающую падение нескольких звезд одновременно.
Создайте приложение, отображающее хаотичное движение звезды в окне.
Разработайте программу анимации взлета ракеты. Старт осу- ществляется по нажатию специальной «красной» кнопки.
Создайте программу, показывающую движение окружности вдоль многоугольника. Число вершин вводится пользователем до анимации.
Создайте приложение, отображающее броуновское движение молекулы в окне.
Разработайте программу анимации движения планет в Солнеч- ной системе.
Создайте программу, показывающую движение квадратика по траектории, состоящей из 100 точек, хранящихся в специальном массиве.
Создайте приложение, имитирующие механические часы.
Разработайте программу анимации падения нескольких лист- ков с дерева. Движение не должно быть линейным.
Создайте программу, показывающую движение окружности по спирали с плавно изменяющейся скоростью.
Создайте приложение, отображающее движение автомобиля с вращающимися колесами.

МЕТОДЫ

Цель практической работы: научиться работать с методами, на- писать программу с использованием методов.

Общие понятия

Метод – это элемент класса, который содержит программный код.
Метод имеет следующую структуру:

[атрибуты] [спецификторы] тип имя ([параметры])
{
Тело метода;
}

Атрибуты – это особые указания компилятору на свойства метода.
Атрибуты используются редко.
Спецификаторы – это ключевые слова, предназначенные для раз- ных 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, кото- рый 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 ре- зультат 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 пере- менной. Параметры разделяются запятой.
Тело метода – это обычный программный код, за исключением то- го, что он не может содержать определения других методов, классов, пространств имен и т. д. Если метод должен возвращать какой-то результат, то обязательно в конце должно присутствовать ключевое слово return с возвращаемым значением. Если возвращение результатов не нужно, то использование ключевого слова return не обязательно, хотя и допускается.
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);
}
}
```

Перегрузка методов

Язык C# позволяет создавать несколько методов с одинаковыми име- нами, но разными параметрами. Компилятор автоматически подберет наиболее подходящий метод при построении программы. Например, мож- но 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 па- раметрами, а во втором – с вещественными:
```
Pow(3, 17);
Pow(3.0, 17.0);
```
Параметры по умолчанию
Язык C# начиная с версии 4.0 (Visual Studio 2010), позволяет зада- вать 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 бе- рет значение по умолчанию.
Параметры по умолчанию можно ставить только в правой части списка параметров, например, такая сигнатура метода компилятором принята не будет:
```
private void GetData(int Optional = 5, int Number)
```
Передача параметров по значению и по ссылке
Когда параметры передаются в метод обычным образом (без до- полнительных ключевых слов ref и out), любые изменения параметров внутри метода не влияют на его значение в основной программе. Пред- положим, у нас есть следующий метод:
```
private void Calc(int Number)
{
Number = 10;
}
```
Видно,что внутриметодапроисходит 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 пере- менной 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 на- зывается *передачей по значению*.
Чтобы метод мог изменять переданную ему переменную, ее следу- ет передавать с ключевым словом ref– оно должно быть как в сигнату- ре 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 на- зывается *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 сло- вом 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

Написать метод min(x, y), находящий минимальное значение из двух чисел. С его помощью найти минимальное значение из четырех чисел a, b, c, d.

Написать метод max(x, y), находящий максимальное значение из двух чисел. С его помощью найти максимальное значение из четырех чисел a, b, c, d.

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

$$10 \quad \frac{i}{x^i}.$$

$$i \square 1$$

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

$$10 \quad i$$

$$\square \; i \cdot$$

$$i \square 1 \qquad \overline{\overline{x}}$$

Написать метод, вычисляющий значение $x^n/(n+x)$. С его помо- щью вычислить выражение:

$$10 \quad x^i$$

$$\square \quad .$$

$$i \square 1 \qquad \overline{\overline{x+i}}$$

Написать метод, вычисляющий значение sin(x) + cos(2 * x). С его помощью определить, в какой из точек a, b или c значение будет минимальным.

Написать метод, вычисляющий значение $x^2 + y^2$. С его помо- щью определить, с какой парой чисел (a, b) или (c, d) значение будет максимальным.

Написать метод, вычисляющий значение $x^2 * y^3 * \sqrt{z}$. С его по- мощью определить, с какой тройкой чисел (a, b, c) или (d, e, f) значение будет максимальным.

Написать метод, который у четных чисел меняет знак, а нечет- ные числа оставляет без изменения. С его помощью обработать ряд чи- сел от 1 до 10.

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

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

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

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

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.

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 обра- ботать пять разных строк и отобразить результаты на экране.

Написать метод, который подсчитывает количество знаков препинания в переданной строке. С его помощью обработать пять раз- ных строк и отобразить результаты на экране.
Написать метод, который находит сумму чисел в переданной строке. Числом считается непрерывная последовательность цифр, отде- ленная от остального текста пробелами или расположенная в начале ли- бо конце строки. Допустимо использовать метод Split класса String. С помощью этого метода обработать пять разных строк и отобразить результаты на экране.
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 помо- щью этого метода обработать пять разных строк и отобразить результа- ты на экране.
Написать метод, находящий сумму матриц одинакового разме- ра и возвращающий новую матрицу. С помощью этого метода обрабо- тать пары матриц и отобразить результаты на экране.
Написать метод, находящий сумму элементов, находящихся не на главной диагонали переданной матрицы. С помощью этого метода обработать пары матриц и отобразить результаты на экране.

РЕКУРСИЯ

Цель практической работы: изучить рекурсивные методы, напи- сать программу с использованием рекурсии.

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 $0! = 1! = 1$, $n! = 1*2*3…*n$. 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 факто- риала. 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 обо- значим 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 вы- числений получается, что F(3) = 3 * 2 * 1.

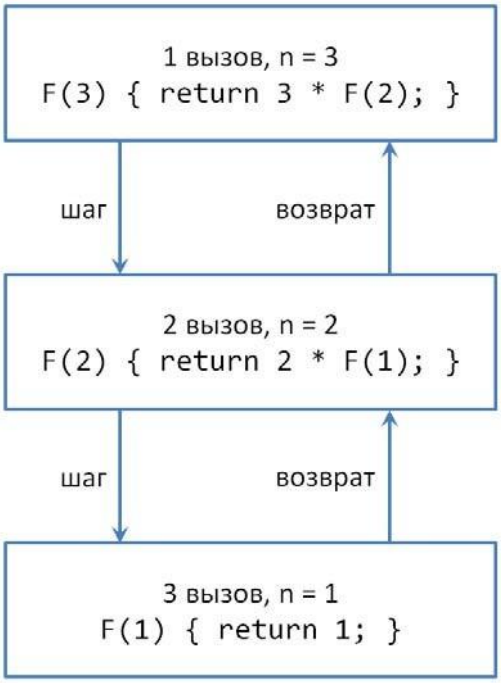


Рис. 14.1. Структура рекурсивных вызовов

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 дан- ного метода с другими параметрами.

Что необходимо знать для реализации рекурсивного процесса? Со входом в рекурсию осуществляется вызов метода, а для выхода не- обходимо 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 за- поминаются не только адреса точек возврата, но и копии значений всех параметров. По этим копиям восстанавливается при возврате вызываю- щий метод. При развертывании рекурсии за счет создания копий пара-

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

Следует понимать, что любой рекурсивный метод можно преобра- зовать в обычный метод с использованием циклов. И практически лю- бой метод можно преобразовать в рекурсивный, если выявить рекур- рентное соотношение между вычисляемыми в методе значениями.

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

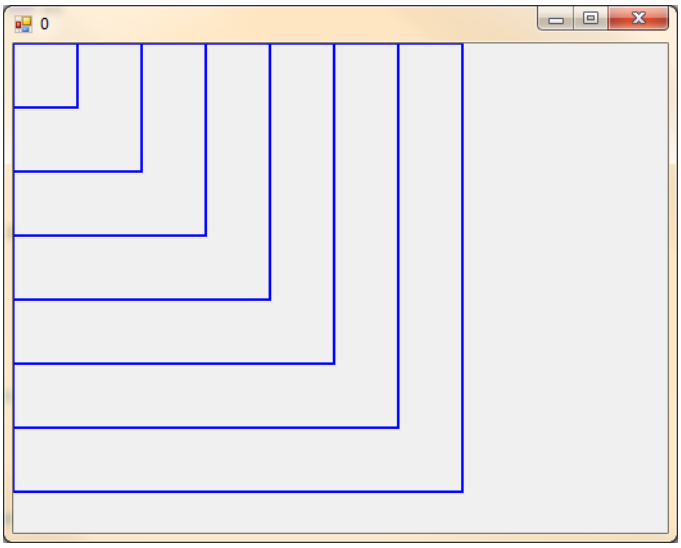


Рис. 14.2. Набор квадратов

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 неизмен- ны и находятся в точке (0, 0). Поэтому в параметрах метода MyDraw дос- таточно передавать *x* и *y* для правого нижнего угла. Также в параметрах передается *N*, значение которой определяет текущую вложенность ре- курсии (сколько вызовов рекурсии еще будет).

Формирование задержки с помощью таймера

Графические конструкции иногда требуется рассматривать дина- мически в процессе их построения. Поэтому зачастую используется та- кая схема вывода графики:

Вывод графического элемента.

Задержка на n миллисекунд.

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;
}
```

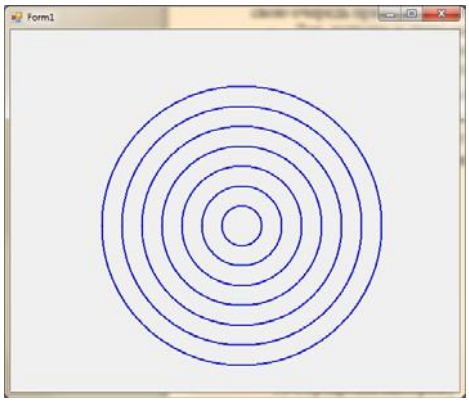
Идея данного подхода заключается в организации бесконечного цик- ла, который будет проверять значение некого флага. Однако значение фла- га может измениться при наступлении события Tick таймера, то есть через заданный в таймере промежуток времени. Однако бесконечный цикл, опи- санный выше, останется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

Перед выполнением индивидуального задания по практической ра- боте разработайте приложение, строящее ряд увеличивающихся квадра- тов (рис. 14.2). Квадраты выводятся последовательно через одну секунду.

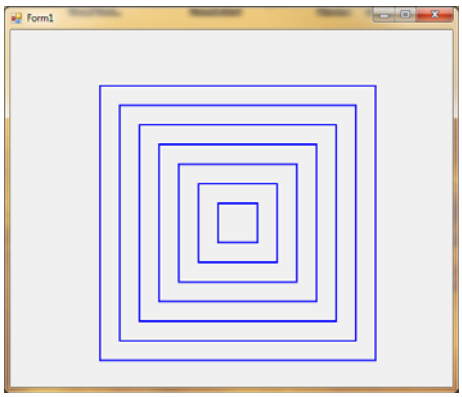
Индивидуальное задание

Напишите приложение, которое строит ряд окружностей. Центр окружностей совпадает с центром экрана. Число окружностей за- дается при первом вызове рекурсивного метода.

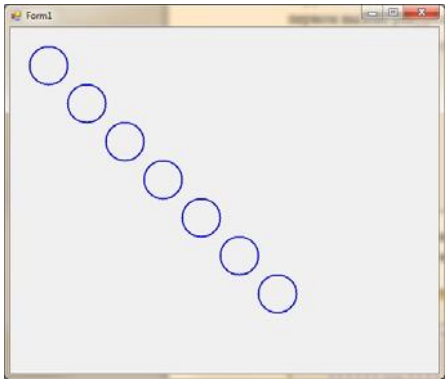
Напишите приложение, которое строит ряд квадратов. Центр квадратов совпадает с



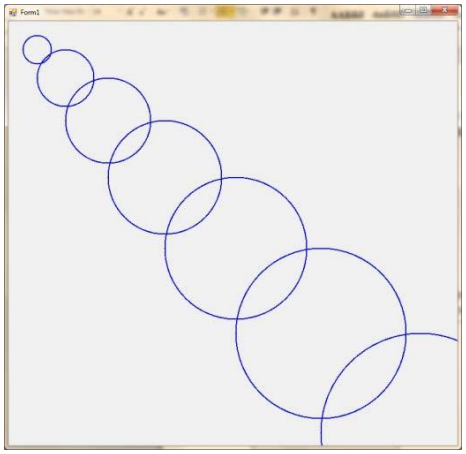
центром экрана. Число квадратов задается при первом вызове рекурсивного метода.



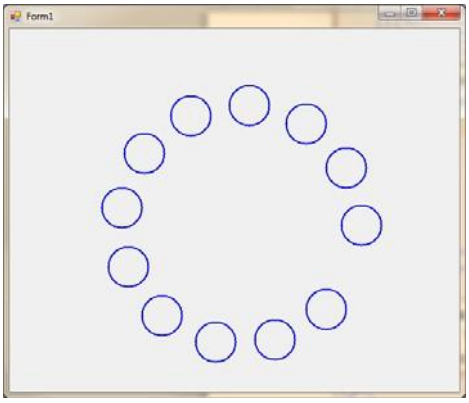
Напишите приложение, которое строит ряд окружностей по диагонали. Число окружностей задается при первом вызове рекур- сивного метода.



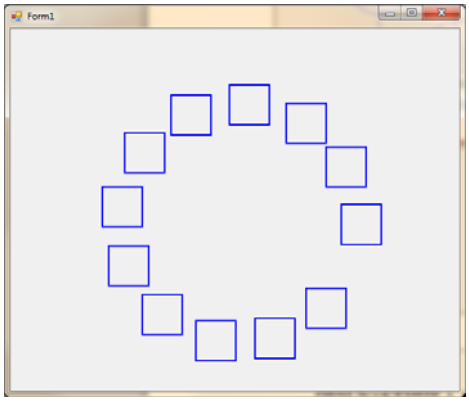
Напишите приложение, которое строит ряд увеличивающихся окружностей по диагонали. Число окружностей задается при первом вызове рекурсивного метода.



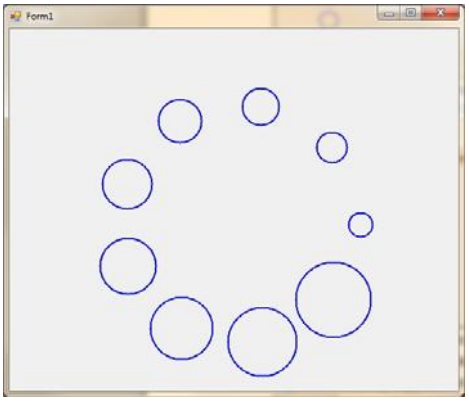
Напишите приложение, которое строит ряд окружностей, цен- тры которых лежат на окружности. Число окружностей задается при первом вызове рекурсивного метода.



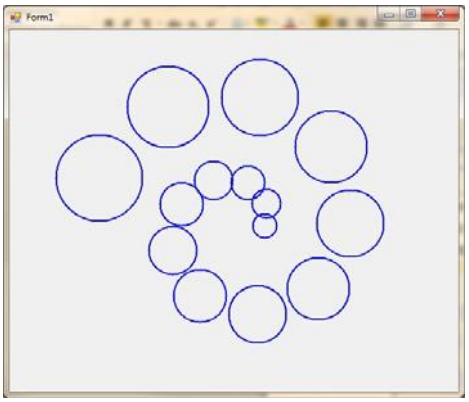
Напишите приложение, которое строит ряд квадратов, центры которых лежат на окружности. Число квадратов задается при первом вызове рекурсивного метода.



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



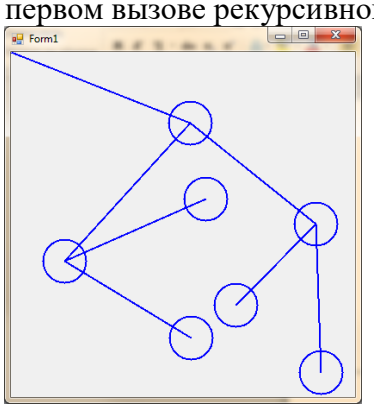
Напишите приложение, которое строит ряд увеличивающихся окружностей, центры которых лежат на спирали. Число окружностей задается при первом вызове рекурсивного метода.



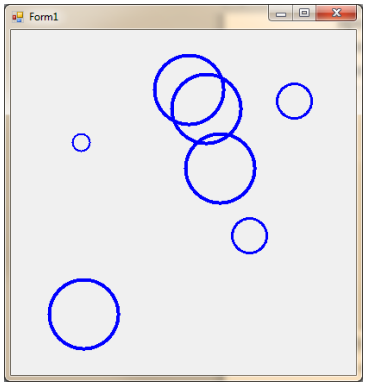
Вычислить, используя рекурсию, выражение:

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

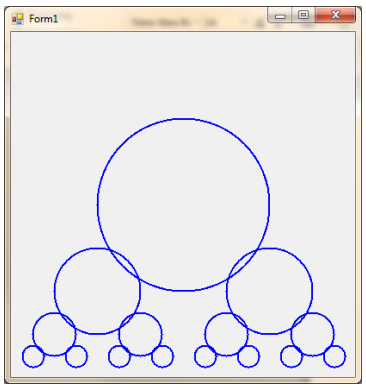
Напишите приложение, которое строит ряд окружностей. Чис- ло окружностей удваивается на каждом шаге (в рекурсивном методе происходит два рекурсивных вызова). Центры окружностей выбираются каждый раз произвольно (случайно). Линии связывают центры окруж- ностей «предка» и «порожденных» от нее. Число рекурсий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



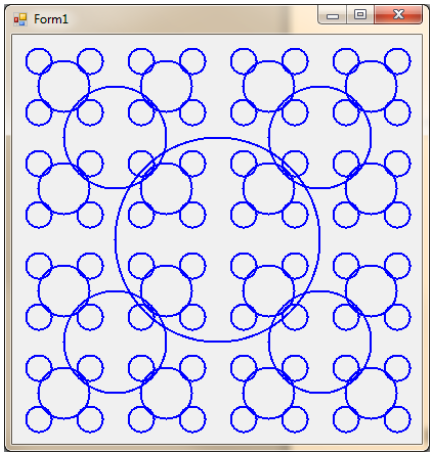
Напишите приложение, которое строит ряд увеличивающихся ок- ружностей. Число окружностей удваивается на каждом шаге (в рекурсивном методе происходит два рекурсивных вызова). Центры окружностей выбира- ются каждый раз произвольно (случайно). Толщина линий также увеличива- ется. Число рекурсий задается при первом вызове рекурсивного метода.



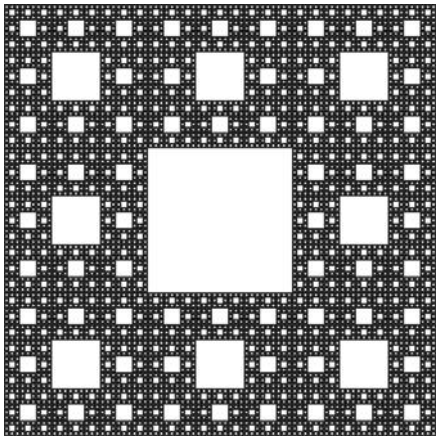
Напишите приложение, которое строит ряд уменьшающихся окружностей. Число окружностей удваивается на каждом шаге (в рекур- сивном методе происходит два рекурсивных вызова). Число рекурсий задается при первом вызове рекурсивного метода.



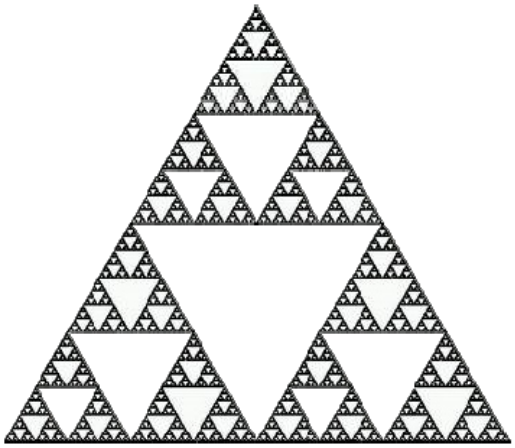
Напишите приложение, которое строит приведенное ниже изо- бражение. Число рекурсий задается при первом вызове рекурсивного метода. На каждом шаге число окружностей увеличивается в четыре раза (в рекурсивном методе происходит четыре рекурсивных вызова).



Постройте ковер Серпинского.



Разработайте программу построения треугольника Серпинского.



Реализуйте программу визуализации построения первых n ша- гов множества Кантора.
Реализуйте рекурсивный алгоритм вычисления n-го числа Фи- боначчи.
Реализуйте рекурсивный алгоритм вычисления n-го факториала.
Реализуйте рекурсивный подсчет суммы всех элементов масси- ва. Сумма элементов массива считается по следующему алгоритму: массив делится пополам, подсчитываются и складываются суммы эле- ментов в каждой половине. Сумма элементов в половине массива под- считывается по тому же алгоритму, то есть снова путем деления попо- лам. Деления происходят, пока в получившихся кусках массива не окажется по одному элементу и вычисление суммы, соответственно, не станет тривиальным.
Дана монотонная последовательность, в которой каждое нату- ральное 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 последова- тельности.

Проверка выполнения самостоятельной работы. Самостоятельная работа направлена на самостоятельное освоение и закрепление обучающимися практических умений и знаний, овладение профессиональными компетенциями.

Самостоятельная подготовка обучающихся по дисциплине предполагает следующие виды и формы работы:

- Систематическая проработка конспектов занятий, учебной и специальной технической литературы.
- Самостоятельное изучение материала и конспектирование лекций по учебной и специальной технической литературе.
- Выполнение расчетных заданий.
- Работа со справочной литературой и нормативными материалами.
- Оформление отчетов по лабораторным и практическим работам, и подготовка к их защите.

Проверка выполнения контрольных работ. Контрольная работа проводится с целью контроля усвоенных умений и знаний и последующего анализа типичных ошибок и затруднений обучающихся в конце изучения темы или раздела. Согласно календарно-тематическому плану дисциплины предусмотрено проведение следующих контрольных работ:

- *Контрольная работа №1 по теме «Основы программирования на C#».*

Спецификации контрольных работ приведены ниже в данном комплекте ФОС.

Сводная таблица по применяемым формам и методам текущего контроля и оценки результатов обучения

Результаты обучения	Формы и методы контроля и оценки результатов обучения
Освоенные умения:	
– определять задачи для поиска информации; – определять необходимые источники информации; – планировать процесс поиска; – оформлять результаты поиска; – применять средства информационных технологий для решения профессиональных задач; использовать современное программное обеспечение; – анализировать проектную и техническую документацию; – оценивать размер минимального набора тестов; – разрабатывать тестовые пакеты и тестовые сценарии; – выявлять ошибки в системных компонентах на основе спецификаций; – организовывать заданную интеграцию модулей в программные средства на базе имеющейся архитектуры и автоматизации бизнес-процессов; – выполнять тестирование интеграции; – обучающийся должен уметь применять информационные технологии для выполнения откладки программного модуля в предметных областях.	Выполнение и защита практических работ № 1-52 Решение задач Устный опрос во время занятия, доклад по выбранной теме. Контрольная работа №1 Зачет с оценкой
Усвоенные знания:	
– приемы структурирования информации; – формат оформления результатов поиска информации – современные средства и устройства информатизации; – порядок их применения и программное обеспечение в профессиональной деятельности – модели процесса разработки программного обеспечения; – основные принципы процесса разработки программного обеспечения;	Выполнение и защита практических работ № 1-52 Решение задач Устный опрос во время занятия, тестирование Контрольная работа №1 Зачет с оценкой

– основные подходы к интегрированию программных модулей; – современные технологии и инструменты интеграции; – модели процесса разработки программного обеспечения; – основные принципы процесса разработки программного обеспечения; – основные методы отладки; – методы и схемы обработки исключительных ситуаций; – обучающий должен знать навыки для применения ИТ при решении задач предметной области.	

3.2 Форма промежуточной аттестации

Промежуточная аттестация по дисциплине Программирование в среде Visual Studio – контрольная работа в 4 семестре, дифференцированный зачёт в 5 семестре, *экзамен в 6 семестре*, спецификация которого содержится в данном комплекте ФОС.

Обучающиеся допускаются к сдаче экзамена при выполнении всех видов самостоятельной работы, практических и контрольных работ, предусмотренных рабочей программой и календарно-тематическим планом дисциплины.

Зачет с оценкой итоговая контрольная работа проводится за счет времени, отведенного на изучение дисциплины. При условии своевременного и качественного выполнения обучающимся всех видов работ, предусмотренных рабочей программой дисциплины.

Итоговая контрольная работа

1) Составить программу:

а) вычисления значения функции $y=7x^2+3x+6$ при любом значении x;

2) В трехзначном числе x зачеркнули первую цифру. Когда оставшееся число умножили на 10, а произведение сложили с первой цифрой числа x, то получилось число 564. Найти число x.

3) Вычислить значение логического выражения при следующих значениях логических величин X, Y и Z: X = Ложь, Y = Истина, Z = Ложь:

а) X или Z; б) X и Y; в) X и Z.

4) Известны две скорости: одна в километрах в час, другая — в метрах в секунду. Какая из скоростей больше?

5) Составить программу, которая в зависимости от порядкового номера дня месяца (1, 2, …, 12) выводит на экран его название (январь, февраль, …, декабрь).

6) Дана непустая последовательность неотрицательных целых чисел, оканчивающаяся отрицательным числом. Найти среднее арифметическое всех чисел последовательности (без учета отрицательного числа).

Перечень вопросов к *зачету с оценкой*

1. Что такое Visual Studio? а) Программное обеспечение для графического дизайна б) Инструмент для редактирования видео в) Интегрированная среда разработки (IDE) г) Приложение для 3D-моделирования Ответ: в) Интегрированная среда разработки (IDE)

2. Какие языки программирования поддерживаются Visual Studio? a) C# и Python б) Java и Ruby в) C++ и JavaScript г) Все вышеперечисленное Ответ: г) Все вышеперечисленное

3. Какова основная цель редактора кода Visual Studio? a) Для выполнения кода b) Для отображения документации по коду c) Для написания, редактирования и управления исходным кодом d) Для запуска и отладки приложений Ответ: c) Для написания, редактирования и управления исходным кодом

4. Какая функция Visual Studio обеспечивает завершение кода и подсказки при вводе? a) IntelliSense b) Live Share c) Фрагменты кода d) Интеграция с Git Ответ: a) IntelliSense

5. Что такое решение в Visual Studio? a) Папка, содержащая файлы проекта б) Коллекция связанных проектов в) Исполняемый файл приложения г) Библиотека для дизайна пользовательского интерфейса Ответ: б) Коллекция связанных проектов

6. Какой инструмент Visual Studio используется для выявления и устранения проблем с кодом? a) Обозреватель решений b) Список ошибок c) Список задач d) Инструменты отладки Ответ: b) Список ошибок

7. Для какой системы контроля версий Visual Studio предоставляет встроенную поддержку? a) Subversion (SVN) b) Git c) Mercurial d) Perforce Ответ: b) Git

8.Какова цель немедленного окна в Visual Studio? a) Для просмотра списка ошибок и предупреждений б) Для выполнения SQL-запросов в) Для оценки выражений и выполнения кода во время отладки г) Для просмотра стека вызовов во время отладки Ответ: в) Для оценки выражений и выполнения кода во время отладки

9. Какой инструмент в Visual Studio используется для создания подключений и запросов к базе данных и управления ими? a) SQL Server Management Studio (SSMS) b) Обозреватель базы данных c) Анализатор запросов d) Мастер подключения данных Ответ: b) Обозреватель базы данных

10. Какое расширение в Visual Studio используется для добавления дополнительных функций в IDE? a) Менеджер дополнений b) Реестр плагинов c) Менеджер расширений d) Marketplace Ответ: c) Менеджер расширений

11. Visual Studio Live Share позволяет разработчикам: a) делиться снимками экрана своего кода; б) совместно редактировать код в режиме реального времени с другими разработчиками; в) отслеживать использование ЦП и памяти приложения; г) развертывать приложения в облаке. Ответ: б) Совместное редактирование кода в режиме реального времени с другими разработчиками.

12. Какова цель панели инструментов в Visual Studio? a) Отобразить список ошибок и предупреждений. б) Предоставить набор элементов управления и компонентов для проектирования пользовательского интерфейса. в) Управлять операциями системы контроля версий. г) Просмотреть схему базы данных приложения. Ответ: б) Предоставить набор элементов управления и компонентов. для дизайна пользовательского интерфейса

13. Какая функция Visual Studio позволяет разработчикам анализировать производительность своих приложений? a) Профилировщик б) Отладчик в) IntelliTrace г) Монитор производительности Ответ: a) Профилировщик

14.Какова цель консоли диспетчера пакетов в Visual Studio? a) Для управления установленными расширениями и надстройками. b) Для управления пакетами NuGet и выполнения команд, связанных с пакетами. c) Для выполнения SQL-запросов к базе данных приложения. d) Для просмотра и управления журналами приложения. Ответ: b) Для управления пакетами NuGet и выполнять команды, связанные с пакетом

15. Какова цель функции точки остановки в Visual Studio во время отладки? a) Для остановки выполнения приложения и отображения вызовов стека б) Для приостановки выполнения приложения по коду строки c) Для просмотра журналов приложений и управления ими d) Для пошагового выполнения кода ответа: b) Для приостановки выполнения приложения по коду строки вызова

16. Какой инструмент в Visual Studio используется для управления ссылками и зависимостями проекта? а) Зависимости Диспетчера б) Обозреватель решений в) Консоль диспетчера пакета г) Диспетчер ссылок Ответ: б) Обозреватель решений

17. Для чего предназначен обозреватель тестов в Visual Studio? а) Для управления модульными тестами и просмотра результатов тестов б) Для просмотра журналов приложений и управления ими в) Для выполнения SQL-запросов к базе данных приложения г) Для анализа производительности Ответ приложений: а) Для управления модульными тестами и просмотра результатов тестов

18. Какая функция Visual Studio позволяет разработчикам просматривать иерархию классов и методов в своем коде и перемещаться по ней? а) Обозреватель объектов б) Фрагменты кода в) Диаграмма классов г) Иерархия вызовов Ответ: а) Обозреватель объектов

19. Для чего предназначено окно непосредственно в Visual Studio? а) Для просмотра списка ошибок и предупреждений б) Для выполнения SQL-запросов в) Для расчета выражений и выполнения кода во время отладки г) Для просмотра стека вызовов во время отладки Ответ: в) Для расчета выражений и выполнения кода во время отладки

20. Какой инструмент в Visual Studio используется для ограничения и устранения проблем с кодом? а) Обозреватель решений б) Список ошибок в) Список задач г) Инструменты отладки Ответ: б) Список ошибок

21. Visual Studio обеспечивает встроенную поддержку для какой системы контроля? а) Subversion (SVN) б) Git в) Mercurial г) Волей-неволей Ответ: б) Git

22.Какова цель немедленного окна в Visual Studio? а) Для просмотра списка ошибок и предупреждений б) Для выполнения SQL-запросов в) Для оценки выражений и выполнения кода во время отладки г) Для просмотра стека вызовов во время отладки Ответ: в) Для оценки выражений и выполнения кода во время отладки

23. Какой инструмент в Visual Studio используется для создания подключений и запросов к базе данных и управления ими? a) SQL Server Management Studio (SSMS) b) Обозреватель базы данных c) Анализатор запросов d) Мастер подключения данных Ответ: b) Обозреватель базы данных

24. Какое расширение в Visual Studio используется для добавления дополнительных функций в IDE? a) Менеджер дополнений b) Реестр плагинов c) Менеджер расширений d) Marketplace Ответ: c) Менеджер расширений

25. Visual Studio Live Share позволяет разработчикам: а) делиться снимками экрана своего кода; б) совместно редактировать код в режиме реального времени с другими разработчиками; в) отслеживать использование процессора и памяти приложением; г) развертывать приложения в облаке. Ответ: б) Совместное редактирование кода в режиме реального времени с другими разработчиками.

26. Какова цель панели инструментов в Visual Studio? а) Отобразить список ошибок и предупреждений. б) Предоставить набор элементов управления и компонентов для проектирования пользовательского интерфейса. в) Управлять операциями системы контроля версий. г) Просмотреть схему базы данных приложения. Ответ: б) Предоставить набор элементов управления и компонентов. для дизайна пользовательского интерфейса

27. Какая функция Visual Studio позволяет разработчикам анализировать производительность своих приложений? а) Профилировщик б) Отладчик в) IntelliTrace г) Монитор производительности Ответ: а) Профилировщик

28.Какова цель консоли диспетчера пакетов в Visual Studio? a) Для управления установленными расширениями и надстройками. b) Для управления пакетами NuGet и выполнения команд, связанных с пакетами. c) Для выполнения SQL-запросов к базе данных приложения. d) Для просмотра и управления журналами приложения. Ответ: b) Для управления пакетами NuGet и выполнять команды, связанные с пакетом

29. Какова цель функции точки останова во время отладки в Visual Studio? а) Остановить выполнение приложения и отобразить стек вызовов б) Приостановить

выполнение приложения на определенной строке кода в) Просмотреть журналы приложения и управлять ими г) Пошаговое выполнение кода Ответ: б) Приостановить выполнение приложения в определенной строке кода

30. Какой инструмент в Visual Studio используется для управления ссылками и зависимостями проекта? a) Диспетчер зависимостей b) Обозреватель решений c) Консоль диспетчера пакетов d) Диспетчер ссылок Ответ: b) Обозреватель решений

Перечень вопросов к *экзамену*

1. Основные понятия и определения; единый каркас среды разработки: библиотека классов.
2. Общеязыковая исполнительная среда CLR.
3. Технология .NET. .NET Framework – каркас среды разработки.
4. Виды проектов.
5. Пространство имен: определение и использование.
6. Пространство имен System. Объект. Конструктор.
7. Оператор new.
8. Работа с решением. Создание проекта. Добавление в решение проекта, формы, библиотеки.
9. Исключительные ситуации и события.
10. Окно вывода сообщения.
11. Диалоговые окна.
12. Язык C#. Типы данных; типы значений и ссылочные типы.
13. Язык C#. Переменные. Инициализация переменных по умолчанию.
14. Язык C#. Операции: арифметические, логические.
15. Язык C#. Реализация ветвлений (if, switch). Примеры.
16. Язык C#. Операторы цикла. Примеры.
17. Язык C#. Работа с массивами. Описание одномерных и многомерных массивов. Инициализация массивов.
18. Методы и свойства класса System.Array.
19. Язык C#. Работа с символьными данными.
20. Методы и свойства класса System.Char.
21. Язык C#. Работа со строковыми данными.
22. Методы и свойства класса System.String.
23. Язык C#. Работа с файлами: классы File, FileInfo.
24. Язык C#. Работа с файлами: классы StreamReader, StreamWriter.
25. Язык C#. Классы пользователя. Основные понятия. Описание класса.
26. Язык C#. Классы: поля и константы. Примеры описания.
27. Язык C#. Классы: виды методов. Примеры описания.
28. Элемент «Форма». Свойства формы.
29. Класс Control. Элементы управления. Обзор элементов управления: родители и потомки, фокус, видимость и отклик, расположение и размер, шрифты и цвет.
30. Элементы управления: группа элементов управления (класс GroupBox), метка (класс Label).
31. Кнопки и двоичные переключатели: кнопка (класс Button), флажок (класс CheckBox), переключатель (класс RadioButton).
32. Элемент управления с поддержкой редактирования текста: текстовое поле (класс TextBox), поле ввода с форматированием (класс RichTextBox).
33. Списоки: список (класс ListBox).
34. Поле со списком (класс ComboBox). Класс Panel.
35. Работа с компонентами меню: классы MenuStrip, ContextMenuStrip.
36. Работа с панелями инструментов: классы, ToolStrip, ToolStripContainer.
37. Работа с диалоговыми окнами. Сохранение и открытие файла.

38. Работа с диалоговыми окнами. Установка шрифта, цвета.

4 Система оценивания комплекта ФОС текущего контроля и промежуточной аттестации

Необходимо расписать систему оценивания каждого вида работ.

При оценивании *практической и самостоятельной работы* студента учитывается следующее:

- качество выполнения практической части работы;

- качество оформления отчета по работе;

- качество устных ответов на контрольные вопросы при защите работы.

Каждый вид работы оценивается по пяти бальной шкале.

«5» (отлично) – за глубокое и полное овладение содержанием учебного материала, в котором обучающийся свободно и уверенно ориентируется; за умение практически применять теоретические знания, высказывать и обосновывать свои суждения. Оценка «5» (отлично) предполагает грамотное и логичное изложение ответа.

«4» (хорошо) – если обучающийся полно освоил учебный материал, владеет научно-понятийным аппаратом, ориентируется в изученном материале, осознанно применяет теоретические знания на практике, грамотно излагает ответ, но содержание и форма ответа имеют отдельные неточности.

«3» (удовлетворительно) – если обучающийся обнаруживает знание и понимание основных положений учебного материала, но излагает его неполно, непоследовательно, допускает неточности, в применении теоретических знаний при ответе на практико-ориентированные вопросы; не умеет доказательно обосновать собственные суждения.

«2» (неудовлетворительно) – если обучающийся имеет разрозненные, бессистемные знания, допускает ошибки в определении базовых понятий, искажает их смысл; не может практически применять теоретические знания.

Тест оценивается по пяти бальной шкале следующим образом: стоимость каждого вопроса 1 балл. За правильный ответ студент получает 1 балл. За неверный ответ или его отсутствие баллы не начисляются.

Оценка «5» соответствует 86% – 100% правильных ответов.

Оценка «4» соответствует 73% – 85% правильных ответов.

Оценка «3» соответствует 53% – 72% правильных ответов.

Оценка «2» соответствует 0% – 52% правильных ответов.

Критерии оценивания контрольной работы

Задание к контрольной работе состоит из шести задач, каждая из которых оценивается максимально оценкой «5» (отлично). По результатам оценивания решения двух задач оценка соответствует средней.

Критерии оценивания ответов по экзаменационным билетам.

Ответ по экзаменационному билету оценивается максимально оценкой «5» (отлично).

Первый вопрос максимально оценивается оценкой «5» (отлично).

Второй вопрос максимально оценивается оценкой «5» (отлично).

Задача оценивается максимально оценкой «5» (отлично).

По результатам оценивания всех трех вопросов оценка соответствует средней.