МИНИСТЕРСТВО НАУКИ И ВЫСШЕГО ОБРАЗОВАНИЯ РОССИЙСКОЙ ФЕДЕРАЦИИ

ФЕДЕРАЛЬНОЕ ГОСУДАРСТВЕННОЕ АВТОНОМНОЕ
ОБРАЗОВАТЕЛЬНОЕ УЧРЕЖДЕНИЕ ВЫСШЕГО ОБРАЗОВАНИЯ
«САМАРСКИЙ НАЦИОНАЛЬНЫЙ ИССЛЕДОВАТЕЛЬСКИЙ
УНИВЕРСИТЕТ ИМЕНИ АКАДЕМИКА С.П. КОРОЛЕВА»
(САМАРСКИЙ УНИВЕРСИТЕТ)

А.А. СТОЛБОВА, О.К. ГОЛОВНИН

ТЕОРЕТИЧЕСКИЕ ОСНОВЫ И ПРАКТИЧЕСКИЕ АСПЕКТЫ ИНФОРМАТИКИ И ПРОГРАММИРОВАНИЯ ДЛЯ РЕШЕНИЯ ЗАДАЧ УПРАВЛЕНИЯ И ОБРАБОТКИ ИНФОРМАЦИИ НА ЯЗЫКЕ C#

Рекомендовано редакционно-издательским советом федерального государственного автономного образовательного учреждения высшего образования «Самарский национальный исследовательский университет имени академика С.П. Королева» в качестве учебного пособия для обучающихся по основной образовательной программе высшего образования по направлению подготовки 09.03.01 Информатика и вычислительная техника

С А М А Р А
Издательство Самарского университета
2019

УДК 004.42
ББК 32.973
С81

Рецензенты: канд. техн. наук, доц. Л.С. Зеленко,
д-р техн. наук, проф. А.В. Иващенко

Столбова, Анастасия Александровна

С81 **Теоретические основы и практические аспекты информатики и программирования для решения задач управления и обработки информации на языке C#**: учеб. пособие / *А.А. Столбова, О.К. Головнин*. – Самара: Изд-во Самарского университета, 2019. – 164 с.: ил.

ISBN 978-5-7883-1432-7

Учебное пособие содержит информацию об основах программирования на языке высокого уровня C#. Рассмотрены такие темы, как общие вопросы информатики, структура программ, синтаксис языка C#, основные типы данных, операторы, массивы, алгоритмизация, принципы объектно-ориентированного проектирования и программирования, графический интерфейс пользователя. По каждой теме представлены примеры программного кода, вопросы для самоконтроля, задания для выполнения лабораторных работ. Даются рекомендации по стилю и технологии программирования, оформлению и документированию программного кода. Пособие содержит теоретический и практический материал, касающийся разработки программ для решения задач управления и обработки информации.

Предназначено для студентов высших учебных заведений, обучающихся по направлению подготовки 09.03.01 Информатика и вычислительная техника. Подготовлено на кафедре информационных систем и технологий.

УДК 004.42
ББК 32.973

ISBN 978-5-7883-1432-7

СОДЕРЖАНИЕ

ВВЕДЕНИЕ

Учебное пособие предназначено для формирования профессиональных знаний и практических навыков в области программирования на языке высокого уровня. Описаны практические аспекты использования языка C# и платформы .NET Framework для разработки и отладки программ при решении конкретных задач управления и обработки информации.

Рассмотрены общие вопросы информатики, структура программ, синтаксис языка C#, основные типы данных, операторы, массивы, алгоритмизация, графический интерфейс пользователя. Рассматривается одна из наиболее перспективных технологий программирования – объектно-ориентированное программирование. Дано представление об основных концепциях объектно-ориентированного программирования – абстрагировании, образовании классов, инкапсуляции, наследовании, полиморфизме. Приведены основные методики создания объектно-ориентированных приложений, схем обмена данными, механизмов обработки ошибочных ситуаций. Изложены концепции эффективной организации исходного кода и самодокументирования, важных для понимания кода программы. Описано использование средств отладки среды программирования и средств пошаговой отладки при выполнении тестирования и поиска ошибок в программах.

Пособие предназначено для студентов высших учебных заведений, обучающихся по направлению 09.03.01 «Информатика и вычислительная техника», а также магистров, аспирантов, научных работников и инженеров, желающих получить базовые навыки программирования при решении инженерных задач.

1 ПРОСТЕЙШИЕ ПРОГРАММЫ

1.1 Создание и запуск проекта. Класс Console

Для того, чтобы начать разработку приложения в Visual Studio необходимо создать решение, состоящее из проектов (рис. 1.1). Проект содержит в себе все файлы исходного кода, значки, изображения, файлы данных и прочие элементы, которые будут скомпилированы в исполняемую программу, а также все остальное, что необходимо для выполнения компиляции. Проект также содержит все параметры компилятора и другие файлы конфигурации, которые могут потребоваться разным службам или компонентам, с которыми программа будет взаимодействовать.

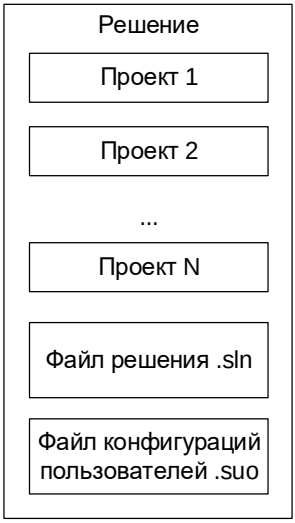

Рис. 1.1. Состав решения

В Visual Studio файл проекта используется обозревателем решений для отображения содержимого и параметров проекта.

Проект содержится в рамках решения, которое может содержать как один, так и несколько проектов вместе с информацией о сборке, параметрами окна Visual Studio и другими файлами. Фактически решение является текстовым файлом, который имеет расширение .sln в собственном уникальном формате.

Также решение включает в себя файл с расширением .suo, в который входят параметры, предпочтения и сведения о конфигурации для каждого пользователя, работавшего над проектом.

Для создания проекта необходимо запустить среду разработки Visual Studio, после чего откроется окно, показанное на рис. 1.2.

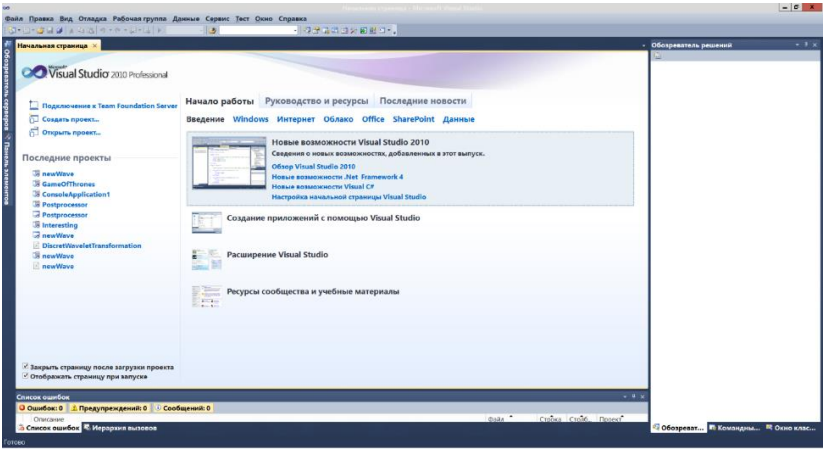

Рис. 1.2. Стартовое окно при запуске Visual Studio

В стартовом окне Visual Studio в пункте меню «Файл» выбрать действие «Создать» и «Проект».

В открывшемся окне выбрать язык (Visual C#), вид приложения (Консольное приложение), задать имя проекта на английском языке, путь к папке, в которой будет храниться решение и имя решения на английском языке и нажать кнопку «OK» (рис. 1.3). В рамках лабораторных работ рекомендуется называть проекты по номерам лабораторных работ, а решение – своей фамилией.

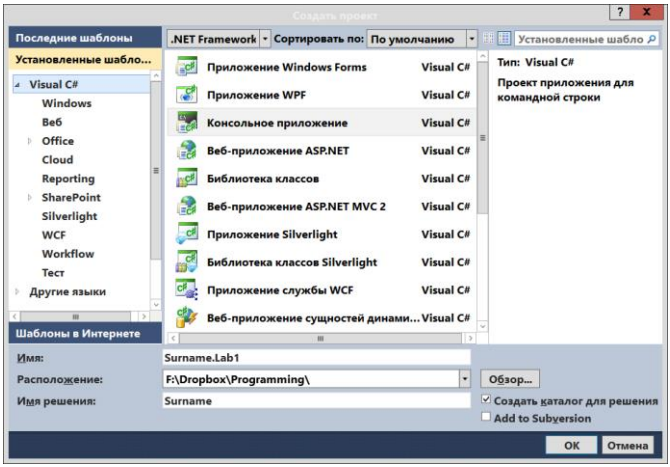

Рис. 1.3. Создание проекта

Перед работой необходимо настроить среду программирования, чтобы были открыты обозреватель решений, список ошибок, окно свойств и панель элементов, как показано на рис. 1.4. Для этого необходимо выбрать соответствующие пункты в меню «Вид». Окно свойств и панель элементов будут необходимы при разработке проектов приложений Windows Forms.

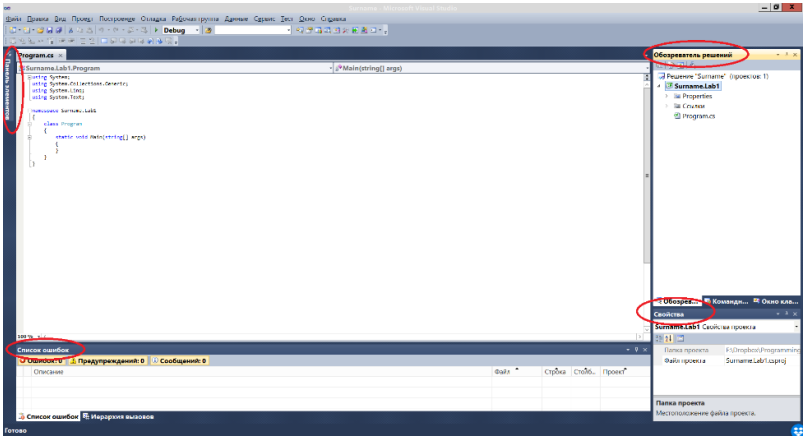

Рис. 1.4. Вид экрана после создания проекта

Напишем программу, которая выводит на экран строку «Hello World!», для чего в главном методе проекта *Main()* вызовем соответствующий метод вывода строки *WriteLine()* класса *Console* (пример 1.1).

Пример 1.1. Вывод строки

```
// Подключаем пространство имен System
using System;

// Объявляем пространство имен Part_1
namespace Surname.Lab1
{
    // Объявляем класс Program
    class Program
    {
        // Описание метода Main
        static void Main(string[] args)
        {
            // Вывод на экран строки «Hello World!»
            Console.WriteLine("Hello World!");
           // Ожидание нажатия любой кнопки
           // препятствует закрытию консоли
            Console.ReadKey();
        }
    }
}
```

Для того, чтобы запустить программу, нужно нажать F5, стрелку или в меню «Отладка» выбрать пункт «Начать отладку».

На рис. 1.5 показан результат выполнения программы.

Рассмотрим подробнее код программы из примера 1.

Директива *using System* позволяет использовать имена стандартных классов из пространства имен *System* без указания его имени.

Ключевое слово *namespace* объявляет для проекта пространство имен, названное Surname.Lab1.

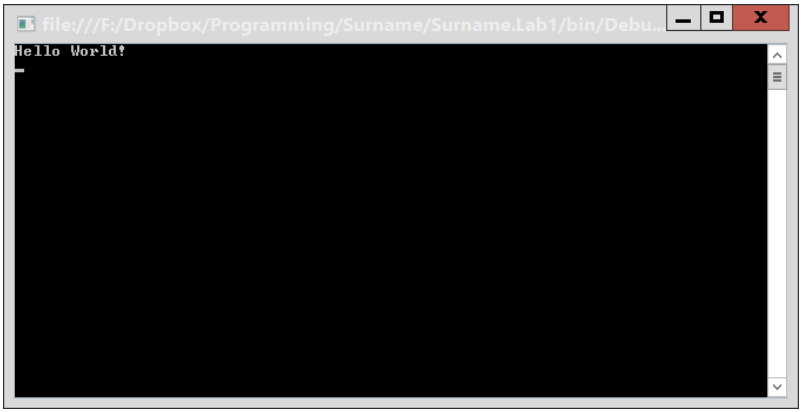

Рис. 1.5. Результат выполнения программы из примера 1

Пространство имен (namespace) – это способ, благодаря которому .NET избегает конфликтов имен между классами. Они предназначены для того, чтобы исключить ситуации, когда вы, например, определяете в проекте класс, представляющий заказчика, называете его Customer, а после этого кто-то другой делает то же самое. Платформа .NET требует, чтобы все имена были объявлены в пределах пространства имен.

Описание класса начинается с ключевого слова *class*.

Ключевые слова – это идентификаторы, которые имеют определенное значение. Они являются зарезервированными.

Идентификатор – это имя какого-либо объекта программы, которое служит для обращения к данному объекту в программе. Идентификаторы могут состоять из одного или нескольких символов, могут начинаться с любой буквы алфавита или знака подчеркивания.

Примеры идентификаторов:

flag, _flag, Flag, FLAG, flag1

Язык C# является регистрозависимым, поэтому прописные и строчные буквы различаются. Так, переменные *flag* и *FLAG* являются

разными. Также идентификаторы не могут начинаться с цифры. Например, имя переменной *4you* является неправильным.

Так как C# является объектно-ориентированным языком, то программа на нем представляет собой совокупность классов.

Класс – это логическая структура, позволяющая создавать свои собственные пользовательские типы путем группирования переменных других типов, методов и событий. С помощью класса описывается некоторая сущность (ее характеристики и возможные действия). Например, класс может описывать студента, автомобиль и т.д.

В программе описывается один класс, названный по умолчанию *Program*.

Класс *Program* содержит метод *Main()* – главный метод, имеющийся в любом приложении, с которого начинается выполнение программы.

Для ввода/вывода данных в консольных приложениях используют класс *Console*, определенный в пространстве имен *System*. В табл. 1.1 приведены методы ввода/вывода класса *Console.*

Таблица 1.1. **Методы ввода/вывода класса *Console***

Метод	Описание
Read()	Читает следующий символ из стандартного входного потока.
ReadLine()	Считывает следующую строку символов из стандартного входного потока.
Write(Boolean)	Записывает текстовое представление заданного логического значения в стандартный выходной поток.
Write(Char)	Записывает значение заданного знака Юникода в стандартный выходной поток.
Write(Char)	Записывает значение заданного знака Юникода в стандартный выходной поток.

Окончание таблицы 1.1

Метод	Описание
Write(Double)	Записывает текстовое представление заданного значения двойной точности с плавающей запятой в стандартный выходной поток.
Write(Int32)	Записывает текстовое представление заданного 32-битового целого числа со знаком в стандартный поток вывода.
Write(String)	Записывает заданное строковое значение в стандартный выходной поток.
WriteLine()	Записывает текущий признак конца строки в стандартный выходной поток.
WriteLine(Boolean)	Записывает текстовое представление заданного логического значения с текущим признаком конца строки в стандартный выходной поток.
WriteLine(Char)	Записывает заданный знак Юникода, за которым следует текущий признак конца строки, в стандартный выходной поток.
WriteLine(Decimal)	Записывает текстовое представление указанного значения Decimal, за которым следует текущий знак завершения строки, в стандартный выходной поток.
WriteLine(Double)	Записывает текстовое представление заданного значения двойной точности с плавающей запятой, за которым следует признак конца строки, в стандартный выходной поток.
WriteLine(Int32)	Записывает текстовое представление заданного 32-битового целого числа со знаком, за которым следует текущий знак завершения строки, в стандартный поток вывода.
WriteLine(String)	Записывает заданное строковое значение, за которым следует текущий признак конца строки, в стандартный выходной поток.

В методе *Main()* у класса *Console* вызывается метод *WriteLine()*, параметром которого является строка “Hello!!!”. Для того, чтобы после запуска и выполнения программы, консоль не закрылась, вызывается метод *ReadLine()* – ожидание ввода символа.

1.2 Типы данных. Операции. Класс *Math*

Для описания данных в языке C# используются **типы**, представленные в табл. 1.2.

Таблица 1.2. **Типы данных**

Тип		Область значений	Размер
Целые типы	sbyte	-128 до 127	Знаковое 8-бит целое
	byte	0 до 255	Беззнаковое 8-бит целое
	short	-32768 до 32767	Знаковое 16-бит целое
	ushort	0 до 65535	Беззнаковое 16-бит целое
	int	-2147483648 до 2147483647	Знаковое 32-бит целое
Целые типы	uint	0 до 4294967295	Беззнаковое 32-бит целое
	long	-9223372036854775808 до 9223372036854775807	Знаковое 64-бит целое
	ulong	0 до 18446744073709551615	Беззнаковое 64-бит целое
Символьный тип	char	U+0000 до U+ffff	16-битовый символ Unicode
Логический тип	bool	true или false	1 байт*

Окончание таблицы 1.2

Тип		Область значений	Размер
Вещественный тип	float	±1,5*10-45 до ±3,4*1033	4 байта, точность – 7 разрядов
	double	±5*10-324 до ±1,7*10306	8 байтов, точность – 16 разрядов
Финансовый тип	decimal	(-7,9 * 1028 до 7,9 * 1028) / (100–28)	16 байт, точность – 28 разрядов
Строковый тип	string	Ограничена объемом доступной памяти	

Переменная – это именованная область памяти, которая предназначена для хранения данных определенного типа. Все переменные, используемые в программе, должны быть объявлены явным образом. Синтаксис объявления переменной выглядит следующим образом:

<Тип данных> <Идентификатор>;

Пример объявления переменной:

```
int b;
```

Инициализировать (задать значение) переменную можно при помощи оператора присваивания (=), в том числе и при объявлении, например:

```
double a = 2.5;
string s = "Hello world!";
```

Значение переменной во время выполнения программы может изменяться.

Операция – это действие над операндами. В качестве операндов могут использоваться переменные. Операции делятся на унарные – над одним операндом, бинарные – над двумя операндами, тернарные – над несколькими операндами. В табл. 1.3 приведены примеры операций.

Таблица 1.3. **Примеры операций**

Вид операции	унарные	бинарные	тернарные
арифметические	- a + a	a + b a - b a * b a / b a % b	
логические	! a	a & b a && b a | b a || b a ^ b	
отношения		a > b a < b a >= b a <= b a == b a != b	
побитовые (поразрядные)	~ a	a & b a | b a ^ b a << b a >> b	
присваивания	++ a a ++ -- a a --	a = b a += b a -= b a *= b a /= b a %= b	

Окончание таблицы 1.3

Вид операции	унарные	бинарные	тернарные
		a &= b a |= b a ^= b a <<= b a >>= b	
условные			a ? b : c

В примере 1.2 показано использование операций присваивания и сложения.

Пример 1.2. Операции

```
int a = 3; // присваивание переменной a значения 3
int b = 5; // присваивание переменной b значения 5

// сложение переменных a и b и присваивание полу-
ченного
  // значения переменной c
int c = a + b;
```

Часто при разработке программ необходимо использовать математические функции: тригонометрические, логарифмические и другие. Для вычисления основных функций определён класс *Math*, который содержит соответствующие методы. Примеры методов, содержащихся в данном классе, приведены в табл. 1.4.

Таблица 1.4. **Некоторые методы класса *Math***

Название	Описание
PI	Константа π
E	Константа e
Abs(Double)	Возвращает абсолютное значение числа двойной точности с плавающей запятой.
Cos(Double)	Возвращает косинус указанного угла в радианах.
Exp(Double)	Возвращает e, возведенное в указанную степень.

Окончание таблицы 1.4

Название	Описание
Log(Double)	Возвращает натуральный логарифм (с основанием e) указанного числа.
Log(Double, Double)	Возвращает логарифм указанного числа в системе счисления с указанным основанием.
Log10(Double)	Возвращает логарифм с основанием 10 указанного числа.
Max(Double, Double)	Возвращает большее из двух чисел двойной точности с плавающей запятой.
Min(Double, Double)	Возвращает меньшее из двух чисел двойной точности с плавающей запятой.
Pow(Double, Double)	Возвращает указанное число, возведенное в указанную степень.
Round(Double)	Округляет заданное число с плавающей запятой двойной точности до ближайшего целого.
Sin(Double)	Возвращает синус указанного угла в радианах.
Sqrt(Double)	Возвращает квадратный корень из указанного числа.
Tan(Double)	Возвращает тангенс указанного угла в радианах.
Truncate(Double)	Вычисляет целую часть заданного числа двойной точности с плавающей запятой.

1.3 Ввод и вывод данных

Важной частью программы является организация диалога с пользователем, реализуемая посредством ввода и вывода данных.

Вывод данных на консоль осуществляется при помощи метода *WriteLine()*. В примере 1.3 приведены способы вывода данных.

Пример 1.3. Вывод данных

```
static void Main()
{
        string str = "one";
        int a = 5;
        Console.WriteLine("one");
```

```
        Console.WriteLine(str);
        Console.WriteLine(a);
        Console.WriteLine(5);
        Console.WriteLine(str + a);
        Console.WriteLine("Число: " + a);
        Console.WriteLine("Число: {0}", a);
        Console.ReadLine();
}
```

Результат выполнения примера 3:

```
one
one
5
5
One5
Число: 5
Число: 5
```

Метод *ReadLine()* позволяет считать с консоли введённую пользователем строку.

Перед тем как считывать что-либо с консоли, необходимо сообщить пользователю, что именно он должен ввести: смысл вводимой информации, тип данных, максимальное и минимальное допустимые значения и т.п.

Пример 1.4. Ввод строки

```
Console.WriteLine("Закрыть приложение? y/n");
string str = Console.ReadLine();
```

В результате выполнения примера 1.4 переменной *str* будет присвоено значение строки, которую введет пользователь.

В классе *Console* определены ввод строки и отдельного символа, но нет метода, который позволяет непосредственно считывать с клавиатуры числа. Для того чтобы преобразовать строковое представление числа к числовому, необходимо воспользоваться методом *Parse()*, имеющимся у числовых типов данных. Ввод числа приведен в примере 1.5.

Пример 1.5. Ввод числа

```
Console.WriteLine("Введите число");
      // Преобразование считанного числа из строкового
   // типа в целочисленый
int number = int.Parse(Console.ReadLine());
```

В результате выполнения примера 5 переменной *number* будет присвоено значение числа, введённого пользователем в строковом формате.

1.4 Линейные программы

Линейной называется программа, все операторы которой выполняются последовательно, в том порядке, в котором они записаны. Это самый простой вид программ.

В примере 1.6 приведена линейная программа, вычисляющая значение следующей функции:

$$z = \sin a + \cos a .$$

Пример 1.6. Вычисление функции

```
using System;
namespace FirstProgram
{
    class Program
```

```
    {
      static void Main()
      {
        // Вывод функции на экран
        Console.WriteLine("Z = Cos(A) + Sin(A)”);
        Console.WriteLine("Введите A: ");
        // Чтение введенных данных и преобразование
        // к числовому типу
       double a = double.Parse(Console.ReadLine());
        // Вычисление значения функции и запись
        // результата в переменную result
        double result = Math.Cos(a) + Math.Sin(a);
        // Вывод результата на экран
        Console.WriteLine("Результат: "+ result);
        Console.ReadKey();
      }
   }
}
```

1.5 Лабораторная работа 1. Простейшие программы

Создать решение для выполнения лабораторных работ.

Создать проект для лабораторной работы 1.

Написать консольную программу, для получения ответа на вопрос и вывода на экран правильного ответа:

1 Пользователь вводит параметры функции.

2 Вывести на экран вопрос «Чему равно значение функции: <функция>?».

3 Получить ответ пользователя.

4 Вывести на экран правильный ответ «Правильный ответ: <ответ>.»

№	Варианты задания
1	$f = \dfrac{\ln^2 b}{\cos\alpha - 1}$
2	$f = \pi\left(\dfrac{\ln b^5}{\sin\alpha + 1}\right)$
3	$f = \dfrac{\sin\alpha + tg(2\alpha)}{\sqrt{\log_3 e^2}}$
4	$f = \dfrac{\cos^7\pi + \sqrt{\ln(b^4)}}{\sin\left(\dfrac{\pi}{2} + \alpha\right)^2}$
5	$f = \sin^2\left(\dfrac{\log_5 b}{\sqrt{\cos(2\alpha)}}\right)$
6	$f = \sin\left(\dfrac{\alpha^3 + \beta^5}{2\pi}\right) + \sqrt[3]{\cos(\alpha + \beta)}$
7	$f = \sin^2\left(\dfrac{3\pi}{4} + \dfrac{\alpha}{3}\right) + \sqrt{\cos\beta}$
8	$f = -4\sin^3(3\alpha) + \dfrac{\sqrt{b}}{\ln(b+2)}$
9	$f = \sqrt{\dfrac{\sin^2\alpha + \cos^3\beta}{\sin^3\alpha - \cos^2\beta}}$
10	$f = \log_b\left(\dfrac{\sqrt{b + \sin\alpha}}{\cos^3\alpha}\right)$

1.6 Вопросы для самоконтроля

1) Из чего состоит решение?
2) Дайте определение пространства имен.
3) Дайте определение идентификаторов.
4) Дайте определение ключевых слов.
5) Дайте определение класса.
6) Какой класс по умолчанию создается по умолчанию в программе?
7) С какого метода начинается выполнение программы в C#?
8) Как нельзя называть переменные в C#?
9) Какие методы класса *Console* предназначены для ввода/вывода данных?
10) Какие типы данных Вы знаете?
11) Какие типы данных являются знаковыми?
12) Какие типы данных являются беззнаковыми?
13) На какие типы делятся операции по числу используемых операндов в них?
14) Какие виды операций можно выполнять над переменными?
15) Какие арифметические операции определены в языке C#?
16) Какие логические операции определены в языке C#?
17) Каким знаком обозначается логические «И»?
18) Каким знаком обозначается логические «ИЛИ»?
19) Объявить переменные следующих типов: вещественный, логический, целый, строковый.
20) Является ли C# регистрозависимым языком?
21) Присвоить значение строковой переменной.
22) Присвоить значение целой переменной.
23) Описать логическую переменную и выполнить инициализацию.
24) Описать вещественную переменную и выполнить инициализацию.

25) Написать код, реализующий вывод на консоль слово «абракадабра».

26) Написать код, реализующий ввод строки с консоли.

27) Какой метод преобразует число в строку?

28) Написать код, реализующий ввод вещественной переменной с консоли.

29) Для чего предназначен класс Math?

30) Какими способами можно вычислить натуральный логарифм, логарифм числа в указанной системе счисления, логарифм с основанием 10?

31) Как получить константу π для вычислений?

32) Дайте определение линейной программы.

2 УСЛОВНЫЕ ОПЕРАТОРЫ

Условные операторы служат для ветвления программы. В зависимости от некоторого условия выполняется тот или другой набор команд.

В C# есть три условных оператора: «if-else», «switch» и «?:» – тернарный оператор.

2.1 Оператор «if-else»

Данный оператор имеет следующий синтаксис:

```
if (условие)
    оператор1;
else
    оператор2;
```

где *условие* – условное выражение, содержащее операции отношения;

оператор1, оператор2 – один или несколько операторов.

Для того, чтобы выполнился *оператор1* необходимо, чтобы выполнилось условие, т.е. приняло значение *true*. В случае, если условие не выполняется (*false)*, выполнится *оператор2*.

Часть *else* является не обязательной и может отсутствовать, тогда оператор будет иметь следующий синтаксис:

```
if (условие)
    оператор1;
```

Напишем программу, определяющую знак числа. В примере 2.1 выполнится оператор, находящийся в части *if*, а в примере 2.2 – оператор, находящийся в части *else*.

Пример 2.1. Выполнение части *if*

```
static void Main()
{
    int number = 10;
    if (number >= 0)
        Console.WriteLine("Число положительное.");
    else
        Console.WriteLine("Число отрицательное.");
}
```

Пример 2.2. Выполнение части *else*

```
static void Main()
{
    int number = -10;
    if (number >= 0)
        Console.WriteLine("Число положительное.");
    else
    Console.WriteLine("Число отрицательное.");
}
```

Если необходимо выполнить несколько операторов, то для их объединения используются фигурные скобки.

Пример 2.3. Написать программу, которая удваивает число, если оно положительное и выводит результат на консоль, а если число отрицательное – ничего не делает.

```
static void Main()
{
```

```
        int number = 10;
        if (number >= 0)
        {
            // блок из 2-х операторов
            int result = number * 2;
            Console.WriteLine("Ответ: " + result);
         }
         else
             Console.WriteLine("Число отрицательное");
         Console.ReadKey();
    }
```

Операторы могут быть любого типа, включая другой оператор *if*, вложенный в исходный оператор *if*. Во вложенных операторах *if* каждое предложение *else* относится к последнему оператору *if*, у которого нет соответствующего объекта *else*.

Пример 2.4. Написать программу, которая определяет, какое из двух введенных чисел больше.

```
    static void Main()
    {
        Console.WriteLine("Введите первое число:");
        int number1 = int.Parse(Console.ReadLine());
        Console.WriteLine("Введите второе число:");
        int number2 = int.Parse(Console.ReadLine());
        if (number1 > number2)
           Console.WriteLine("Первое число больше вто-
рого.");
        else
            // вложенный оператор
            if (number1 < number2)
```

```
                Console.WriteLine("Второе число больше
первого.");
            else
                Console.WriteLine("Числа равны.");
        Console.ReadKey();
    }
```

Логическое выражение может быть составное и состоять из нескольких условий. Для их объединения используются логические бинарные операторы «И», «ИЛИ», «Исключающее ИЛИ», например:

```
    if (number >0 && number % 2 == 0)
        Console.WriteLine("Число {0} чётное и положи-
тельное", number);
```

В табл. 2.1, 2.2 и 2.3 приведены таблицы истинности логических бинарных операторов:

Таблица 2.1. **Таблица истинности логического «И»**

a	**b**	**a && b**
0	0	0
0	1	0
1	0	0
1	1	1

Таблица 2.2. **Таблица истинности логического «ИЛИ»**

a	**b**	**a || b**
0	0	0
0	1	1
1	0	1
1	1	1

Таблица 2.3. **Таблица истинности «исключающего ИЛИ»**

a	b	a ^ b
0	0	0
0	1	1
1	0	1
1	1	0

2.2 Оператор «switch»

Оператор *switch* часто используется вместо оператора *if* при выборе из списка трех или более кандидатов. Он имеет следующий синтаксис:

```
switch (выражение)
{
    case значение1:
        оператор1;
        break;
    case значение2:
        оператор2;
        break;
    default:
        действие;
        break;
}
```

Выражение сравнивается последовательно со значениями. Если выражение равно значению, выполняется соответственный блок кода и при достижении ключевого слова *break* оператор *switch* заканчивает работу. Если выражение не будет соответствовать ни одному значению, тогда выполнится блок после *default*.

Выражение должно возвращать значение следующих типов:

– *char*;
– *string*;
– *bool*;
– целочисленное значение, например, *int* или *long*;
– значение *enum*.

Пример 2.5. Написать программу с использованием оператора *switch*, которая выводит на экран название дня недели в соответствие с вводимым порядковым номером дня:

```
    static void Main()
    {
        Console.WriteLine("Введите порядковый номер
дня недели:");
        int number = int.Parse(Console.ReadLine());
        switch (number)
        {
            case 1:
                Console.WriteLine("Понедельник");
                break;
            case 2:
                Console.WriteLine("Вторник");
                break;
            case 3:
                Console.WriteLine("Среда");
                break;
            case 4:
                Console.WriteLine("Четверг");
                break;
            case 5:
                Console.WriteLine("Пятница");
                break;
```

```
            case 6:
                Console.WriteLine("Суббота");
                break;
            case 7:
                Console.WriteLine("Воскресенье");
                break;
            default:
                Console.WriteLine("Ошибка");
                break;
        }
        Console.ReadKey();
    }
```

2.3 Тернарный оператор «?:»

Этот оператор используется для сокращения объема кода. Им можно заменять простые по сложности операторы *if-else*. Тернарный оператор имеет следующий синтаксис:

```
условие ? оператор1 : оператор2;
```

Сначала вычисляется *условие*. Затем, если оно истинно, то вычисляется *оператор1*, в противном случае – вычисляется *оператор2*.

Пример 2.6. Написать программу с использованием тернарного оператора «?:» для определения знака числа.

```
    static void Main()
    {
        Console.WriteLine("Введите число:");
        int number = int.Parse(Console.ReadLine());
        Console.WriteLine(number >= 0 ? "Число положи-
тельное" : "Число отрицательное");
```

```
        Console.ReadKey();
    }
```

2.4 Контроль ввода. Метод *TryParse*

Метод *TryParse()* аналогичен методу *Parse()*, за исключением того, что метод *TryParse()* не вызывает исключение при сбое преобразования. Отпадает необходимость использования обработки исключений для проверки на наличие *FormatException* (неверный формат входной строки) в случае, если строка является недопустимой и не может быть успешно преобразована.

Синтаксис метода *TryParse()* следующий:

```
    public static bool TryParse(string s, out int re-
sult)
    public static bool TryParse(string s, out double
result)
    public static bool TryParse(string s, out float
result)
```

где *s* – это строка, содержащая преобразуемое число;

result – возвращаемый параметр, который содержит значение, эквивалентное вводимому пользователем строковому представлению числа. Этот параметр передается неинициализированным; любое значение, первоначально предоставленное в объекте *result*, будет перезаписано.

Метод *TryParse()* возвращает значение, указывающее, успешно ли выполнена операция преобразования: *true*, если строка успешно преобразована; в противном случае – значение *false*.

Рассмотрим пример реализации контроля ввода числа. Объявим переменную, в которую нужно записать число и попросим пользователя ввести число. Далее организуем цикл, в котором и будет осу-

ществляться проверка: до тех пор, пока не будет введена корректная строка, просим пользователя ввести число (пример 2.7).

Пример 2.7. Контроль ввода

```
int number;
Console.WriteLine("Введите число: ");
while (!int.TryParse(Console.ReadLine(), out number))
{
    Console.WriteLine("Введите число: ");
}
```

2.5 Исключения. Обработка исключений. Конструкция try-catch-finally

Язык C# обеспечивает встроенную поддержку для обработки нестандартных ситуаций, называемых исключениями, которые могут происходить во время выполнения программы.

Исключениями, или исключительными ситуациями, обычно называются аномалии, которые могут возникать во время выполнения и которые трудно, а порой и вообще невозможно, предусмотреть во время программирования приложения. К числу таких возможных исключений относятся, например, обращение к файлу, которого не существует (*FileNotFoundException*), арифметическое переполнение (*OverflowException*) и др.

Для того чтобы справиться с возможными ошибочными ситуациями в коде C# используется конструкция «try-catch-finally», которая имеет следующий синтаксис:

```
try
{
// Блок кода, предназначенный для обработки ошибок.
```

```
}
catch (ExceptionTypel ex1)
{
    // Обработчик исключения типа ExceptionTypel
}
catch (ExceptionType2 ex2)
{
    // Обработчик исключения типа ExceptionType2.
}
finally
{
    // Код завершения обработки исключений.
}
```

– блок *try* инкапсулируют код, формирующий часть нормальных действий программы, которые потенциально могут столкнуться с серьезными ошибочными ситуациями;

– блок *catch* инкапсулируют код, который обрабатывает ошибочные ситуации, происходящие в коде блока *try*;

– блок *finally* инкапсулируют код, очищающий любые ресурсы или выполняющий другие действия, которые обычно нужно выполнить в конце блоков *try* или *catch*. Важно понимать, что этот блок выполняется независимо от того, было ли сгенерировано исключение. Данный блок является необязательным и может отсутствовать.

При использовании конструкции «try-catch-finally» вначале выполняются все инструкции между операторами *try* и *catch*. Если между этими операторами вдруг возникает исключение, то обычный порядок выполнения останавливается и переходит к инструкции *catch*.

Пример 2.8. Применение конструкции «try-catch-finally»

```
static void Main()
{
```

```
        int x = int.Parse(Console.ReadLine());
        int y = int.Parse(Console.ReadLine());
        try
        {
            int result = x / y;
        }
        catch (DivideByZeroException)
        {
            Console.WriteLine("Деление на ноль!");
        }
        // Блок finally будет выполняться всегда
        finally
        {
            Console.WriteLine("Конец программы");
        }
        Console.ReadLine();
    }
```

2.6 Лабораторная работа 2. Условные операторы

В ранее созданное решение добавить новый проект для лабораторной работы 2.

Написать консольную программу, содержащую лабораторную работу 1 с дополнениями.

Сравнить ответ пользователя с вычисленным результатом и сообщить пользователю результат (ответ верный, ответ неверный). Для выполнения сравнения ответ округлить.

Реализовать меню для выбора действий:

1 – Отгадай ответ

2 – Об авторе (Фамилия И.О., группа)

3 – Выход

При выборе пункта меню 3 запрашивать подтверждение на выход из программы (при нажатии «д» выйти из программы, при нажатии «н» – остаться в программе).

Реализовать проверку на корректность ввода: сообщать об ошибке, если пользователь вводит буквы вместо цифр.

Предусмотреть корректную работу программы в случае деления на 0, взятия корня из отрицательного числа и другие возможные исключительные ситуации (в зависимости от варианта).

Разработать тестовые примеры для проверки исключительных ситуаций.

2.7 Вопросы для самоконтроля

1) Какие условные операторы существуют в языке C#?
2) Для чего нужны условные операторы?
3) Для чего нужен метод TryParse()? Синтаксис.
4) Опишите работу метода TryParse()?
5) В чем отличие метода TryParse() от метода Parse()?
6) Дайте определение исключениям.
7) Какие способы обработки исключений используются в C#?
8) Опишите работу оператора try-catch-finally.
9) Является ли обязательным блок cath?
10) Является ли обязательным блок finally?
11) Могут ли условные выражения быть составными? Приведите пример.
12) Напишите таблицу истинности для логического «И».
13) Напишите таблицу истинности для логического «ИЛИ».
14) Напишите таблицу истинности для исключающего «ИЛИ».
15) *В чем отличие сокращенных логических «И» и «ИЛИ» от обычных?
16) В каких случаях используется оператор if-else?
17) Опишите работу оператора if-else, синтаксис.

18) Является ли блок else обязательным в операторе if-else?

19) В каких случаях используется оператор switch?

20) Опишите работу оператора switch, синтаксис.

21) В каких случаях используется тернарный оператор?

22) Опишите работу тернарного оператора, синтаксис.

23) Какого типа может быть выражение в операторе switch?

24) Для чего нужен блок default оператора switch?

25) С клавиатуры вводятся два числа – количество забитых голов хозяевами и гостями в футбольном матче. Вывести на экран результат игры – победили хозяева/гости/ничья.

26) Пользователь вводит число. Написать программу, проверяющую вводимое число на чётность. Использовать оператор if-else и тернарный оператор.

27) Напишите программу, которая будет проверять число на кратность 3 и 7. Вывести на экран соответствующее сообщение.

28) Подсчитать сумму только положительных чисел из трех заданных.

29) Написать программу, в которой пользователь выбирает цвет консоли (использовать оператор switch).

30) Написать программу для определения, можно ли из отрезков с длинами a, b, c построить треугольник. Любая сторона треугольника меньше суммы двух других сторон и больше их разности ($a < b + c,\ a > b - c;\ b < a + c,\ b > a - c;\ c < a + b,\ c > a - b$).

3 ОПЕРАТОРЫ ЦИКЛА

Циклы являются управляющими конструкциями, позволяя в зависимости от определенных условий выполнять некоторое действие множество раз. В C# имеются следующие виды циклов:

- *for*;
- *foreach*;
- *while*;
- *do-while*.

3.1 Цикл for

Цикл for рекомендуется использовать в тех случаях, когда заранее известно, сколько раз необходимо повторить действия. Синтаксис цикла следующий:

```
    for ([инициализация счетчика]; [условие]; [измене-
ние счетчика])
    {
        //тело цикла
    }
```

При *инициализации счетчика*, как правило, объявляется и инициализируется переменная, которая действует как счетчик для управления циклом. Отметим, что в качестве счетчика могут выступать переменные целого типа. Использование вещественных переменных не рекомендуется.

Раздел *условие* – это выражение условия, имеющее тип *bool*, от истинности которого (при проверке измененного значения счетчика)

зависит, будет ли выполнен еще один проход цикла. Это выражение проверяется перед каждой итерацией цикла, поэтому тело цикла может не выполнить ни одного раза.

Раздел цикла *изменение счетчика* – это выражение, определяющее величину (шаг), на которую будет изменяться значение счетчика при каждом повторении цикла. Отметим, что значение счетчика можно как увеличивать, так и уменьшать.

Эти три основных элемента цикла должны быть разделены точкой с запятой. Цикл *for* повторяется до тех пор, пока в результате проверки выражения условия будет возвращаться значение *true*. Если будет возвращено значение *false*, цикл завершится и управление программой будет передано оператору, следующему за циклом *for*.

Тело цикла является оператором или блоком операторов.

В примере 3.1 показано использование цикла *for*.

Пример 3.1. Написать программу для возведения в квадрат цифр от 1 до 9 (цикл *for*).

```
    static void Main()
    {
        for (int num = 1; num < 10; num++)
        {
            double pow = Math.Pow(num, 2);
            Console.WriteLine("{0} в квадрате равно
{1}.", num, pow);
        }
        Console.ReadKey();
    }
```

В первой части объявления цикла создается и инициализируется счетчик *num* с начальным значением 1. Цикл будет выполняться до тех пор, пока счетчик *num* не станет равен 10. В третьей части значение счетчика увеличивается на 1. Тело цикла представляет собой

блок операторов: возведение числа в квадрат и вывод результата на консоль. Таким образом, цикл выполнится 9 раз.

Все три раздела цикла являются необязательными. Например, следующим образом можно организовать бесконечный цикл:

```
for (; ; )
{
    // Тело цикла
}
```

3.2 Цикл while

Цикл *while* также широко применяется в C#-программах. Его синтаксис выглядит следующим образом:

```
while (условие)
{
    //тело цикла
}
```

Аналогично с циклом *for*, условие является логическим выражением, принимающим значение типа *bool*, а тело цикла является оператором или блоком операторов. Оператор выполняется до тех пор, пока условие принимает значение *true*. Если же условие принимает значение *false*, управление программой передается оператору, который следует сразу за циклом.

Как и в цикле *for*, в цикле *while* условное выражение проверяется до начала цикла, что означает, что код в теле цикла может вообще не выполняться.

В примере 3.2 показано использование цикла while для задачи из примера 3.1.

Пример 3.2. Написать программу для возведения в квадрат цифр от 1 до 9 (цикл *while*).

```
    static void Main()
    {
        int num = 1;
        while (num < 10)
        {
            double pow = Math.Pow(num, 2);
            Console.WriteLine("{0} в квадрате равно
{1}.", num, pow);
            num++;
        }
         Console.ReadKey();
    }
```

3.3 Цикл do-while

В отличие от циклов *for* и *while*, где условие проверяется в самом начале, в цикле *do-while* условие проверяется в конце цикла. Это означает, что цикл *do-while* всегда выполняется как минимум один раз. Синтаксис цикла *do-while* следующий:

```
do
{
    //тело цикла
}
while (условие);
```

При наличии в цикле только одного оператора фигурные скобки не обязательны, но их часто используют для улучшения читабельности конструкции цикла *do-while*, поскольку его легко спутать с циклом *while*. Если при проверке условие принимает значение *true*, цикл *do-while* выполняется еще раз. В примере 3.3 показано применение цикла *do-while*.

Пример 3.3. Написать программу для возведения в квадрат цифр от 1 до 9 (цикл *do-while*).

```
static void Main()
{
    int num = 1;
    do
    {
        double pow = Math.Pow(num, 2);
        Console.WriteLine("{0} в квадрате равно
{1}.", num, pow);
        num++;
    }
    while (num < 10);
    Console.ReadKey();
}
```

3.4 Цикл foreach

Цикл *foreach* служит для циклического обращения к элементам коллекции, представляющей собой группу объектов (например, массивы, списки, перечисления). Ниже приведен синтаксис цикла *foreach*:

```
foreach (тип имя_переменной in коллекция)
{
    //тело цикла
}
```

В данном случае *тип имя_переменной* обозначает тип и имя переменной управления циклом, которая получает значение следующего элемента коллекции на каждом шаге выполнения цикла *foreach*. А коллекция обозначает циклически опрашиваемую коллекцию. Сле-

довательно, тип переменной цикла должен соответствовать типу элемента коллекции. Кроме того, тип может обозначаться ключевым словом *var*. В этом случае компилятор определяет тип переменной цикла, исходя из типа элемента массива.

Оператор цикла *foreach* действует следующим образом. Когда цикл начинается, первый элемент коллекции выбирается и присваивается переменной цикла. На каждом последующем шаге итерации выбирается следующий элемент коллекции, который сохраняется в переменную цикла. Цикл завершается, когда все элементы коллекции окажутся выбранными. В примере 3.4 показано применение цикла *foreach*.

Пример 3.4. Написать программу для вывода на консоль символов строки.

```
static void Main()
{
    string s = "abracadabra";
    foreach (char ch in s)
        Console.WriteLine(ch);
    Console.ReadKey();
}
```

3.5 Лабораторная работа 3. Циклы

В ранее созданное решение добавить новый проект для лабораторной работы 3.

Написать консольную программу, содержащую лабораторную работу 2 с дополнениями.

Зациклить выполнение программы так, чтобы программа закрывалась только при выборе пункта меню «Выход».

В первом пункте меню «Отгадай ответ» реализовать игру: пользователь имеет 3 попытки ввода ответа. Если пользователь ввел пра-

вильный ответ, то сообщить ему о том, что он выиграл и выйти в меню. При неправильном вводе ответа более трёх раз сообщить пользователю о проигрыше и выйти в меню. После каждого ввода ответа сообщать пользователю, сколько попыток ввода у него осталось.

Запрещено использование операторов break, goto, continue для выхода из цикла.

3.6 Вопросы для самоконтроля

1) Для чего используются циклы?
2) Опишите работу и синтаксис цикла *for*.
3) В каких случаях используется цикл *for*?
4) Какого типа должна быть переменная, являющаяся счетчиком для управления циклом?
5) Опишите работу и синтаксис цикла *while*.
6) В каких случаях используется цикл *while*?
7) Опишите работу и синтаксис цикла *do-while*.
8) В каких случаях используется цикл *do-while*?
9) Опишите работу и синтаксис цикла *foreach*.
10) В каких случаях используется цикл *foreach*?
11) Какого типа должно быть условие в циклах?
12) При каком значении условного выражения будет выполняться тело цикла?
13) Что представляет собой тело цикла?
14) Как организовать бесконечный цикл, используя операторы *for, while, do-while*.
15) Чем отличаются циклы *while* и *do-while*?
16) Вывести на консоль символы латинского алфавита.
17) Составьте программу, которая вычисляет сумму чисел от 1 до n. Значение n вводится с клавиатуры.
18) Напечатать те из двузначных чисел, которые делятся на 4, но не делятся на 6.

19) Найти произведение двузначных нечетных чисел кратных 13.

20) Написать программу, вычисляющую факториал числа.

21) Вычислить: (1+2)*(1+2+3)*...*(1+2+...+10).

4 ОДНОМЕРНЫЕ МАССИВЫ

4.1 Объявление и инициализация одномерных массивов

Массивом называют упорядоченную совокупность элементов одного типа. Каждый элемент массива имеет индексы, определяющие порядок элементов. Индексы задаются целочисленным типом. Массивы в C# являются переменными ссылочного типа. В переменных ссылочных типов хранятся ссылки на их данные (объекты). Две переменные ссылочного типа могут ссылаться на один и тот же объект, поэтому операции над одной переменной могут затрагивать объект, на который ссылается другая переменная. В C# массивы могут быть как одномерными, так и многомерными.

Для того, чтобы использовать массив, нужно объявить переменную, с помощью которой будет осуществляться доступ к элементам, и создать экземпляр массива, используя оператор *new*. Синтаксис объявления и создания одномерного массива следующий:

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

Тип элементов массива может быть любым, включая массив. Оператор *new* выделяет память в «куче». Под размером понимается число элементов массива.

До тех пор, пока элементы массива не проинициализированы, они будут принимать значения по умолчанию для заданного типа. Значения по умолчанию для различных типов приведены в табл. 4.1.

В следующем примере создается массив из 5 элементов типа *int*:

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

Таблица 4.1. **Значения по умолчанию**

Тип значения	Значение по умолчанию
bool	false
byte	0
char	'\0'
decimal	0M
double	0.0D
float	0.0F
int	0
long	0L
sbyte	0
short	0
uint	0
ulong	0
ushort	0

В данном примере создается переменная с именем *array*, которая содержит ссылку на массив, и выделяется память под 5 элементов массива типа *int*, значения которых по умолчанию равны 0 (рис. 4.1).

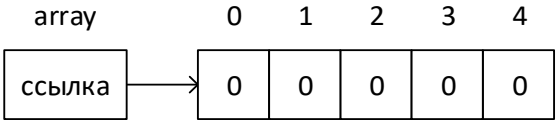

Рис. 4.1. Представление массива

В том случае, если не выделить память под элементы массива (оператор *new*), будет создана переменная со ссылкой, имеющей неопределённое значение *null* (рис. 4.2):

```
int[] array;
```

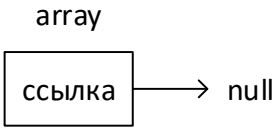

Рис. 4.2. Представление массива без создания его экземпляра

Доступ к элементам одномерного массива осуществляется с помощью индекса. Индекс определяет положение элемента в массиве. Индексация массивов в C# начинается с нуля: массив с *n* элементами индексируется от 0 до *n-1*.

Пример доступа к элементам массива:

```
int[] array = new int[5];
// В элемент с индексом 3 записывается число 666
array[3] = 666;
// Ошибка! Индекс выходит за пределы диапазона
array[5] = 8;
```

Массив можно инициализировать при объявлении. Примеры инициализации массива:

```
int[] array = { 1, 3, 5, 7, 9 };
string[] weekDays = { "Sun", "Mon", "Tue", "Wed",
"Thu", "Fri", "Sat" };
```

4.2 Методы ввода и вывода одномерных массивов. Класс Random

Для работы с массивами необходимо уметь их инициализировать и выводить на консоль. Напишем метод, заполняющий массив случайными значениями (пример 4.1).

Пример 4.1. Заполнить массив случайными значениями.

```
static int[] RandomArray(int[] a, int n)
{
    // Создание экземпляра класса Random
    Random rnd = new Random();
    // Цикл, проходящий по всем элементам массива
    for (int i = 0; i < n; i++)
// Присваивание i-ому элементу массива случайного
// значения
        a[i] = rnd.Next(-10, 10);
    return a;
}
```

Напишем метод вывода массива на консоль, используя цикл *foreach*.

Пример 4.2. Вывести массив на консоль.

```
static void OutputArray(int[] a)
{
    foreach (int i in a)
        Console.Write("{0}   ", i);
}
```

Класс *Random* представляет собой генератор псевдослучайных чисел. Основные методы класса представлены в табл. 4.2.

Таблица 4.2. **Методы класса Random**

Имя	Описание
Next()	Возвращает неотрицательное случайное целое число.
Next(Int32)	Возвращает неотрицательное случайное целое число, которое меньше максимально допустимого значения.
Next(Int32, Int32)	Возвращает случайное целое число в указанном диапазоне.

Окончание таблицы 4.2

NextDouble()	Возвращает случайное число с плавающей запятой, которое больше или равно 0,0 и меньше 1,0.
Sample()	Возвращает случайное число с плавающей запятой в диапазоне от 0,0 до 1,0.

4.3 Класс Array

Класс *Array* предоставляет методы и свойства для создания, изменения, поиска и сортировки массивов, то есть выступает в роли базового класса для всех массивов. Основные свойства и методы класса *Array* представлены в табл. 4.3.

Таблица 4.3. **Основные свойства и методы класса *Array***

Имя	Описание
Свойства	
Length	Получает общее число элементов во всех измерениях массива Array.
Rank	Получает ранг (число измерений) массива Array. Например, одномерный массив возвращает 1, двухмерный массив возвращает 2 и т д
Методы	
IndexOf(Array, Object)	Выполняет поиск указанного объекта внутри всего одномерного массива и возвращает индекс его первого вхождения.
IndexOf(Array, Object, Int32)	Выполняет поиск указанного объекта в диапазоне элементов одномерного массива и возвращает индекс первого найденного совпадения. Диапазон расширяется от указанного индекса до конца массива.
IndexOf(Array, Object, Int32, Int32)	Выполняет поиск указанного объекта в диапазоне элементов одномерного массива и возвращает

Окончание таблицы 4.3

Имя	Описание
	индекс первого найденного совпадения. Диапазон расширяется от указанного индекса заданного числа элементов.
LastIndexOf(Array, Object)	Выполняет поиск заданного объекта и возвращает индекс его последнего вхождения внутри всего одномерного массива Array.
LastIndexOf(Array, Object, Int32)	Выполняет поиск указанного объекта и возвращает индекс его последнего вхождения в диапазоне элементов одномерного массива Array, который начинается с первого элемента и заканчивается элементом с заданным индексом.
LastIndexOf(Array, Object, Int32, Int32)	Выполняет поиск указанного объекта и возвращает индекс последнего вхождения в диапазоне элементов одномерного массива Array, который содержит указанное число элементов и заканчивается элементом с заданным индексом.
Sort(Array)	Сортирует элементы во всем одномерном массиве Array, используя реализацию интерфейса IComparable каждого элемента массива Array.

4.4 Алгоритмы работы с одномерными массивами

4.4.1 Поиск индекса первого вхождения элемента в массив

Необходимо найти индекс первого вхождения элемента *element* в массив *a*, состоящего из *n* элементов.

Алгоритм решения задачи следующий: нужно сравнивать элементы массива с заданным элементом до тех пор, пока не будет найден равный ему. Однако данного условия недостаточно, т.к., если элементов, равных заданному, в массиве нет, произойдет выход за

границы массива (исключение *IndexOutOfRangeEcxeption*). Для решения данной задачи следует использовать цикл *while*:

```
    int i = 0;
    while (i < n && a[i] != element)
        i++;
    Console.WriteLine("Индекс первого вхождения элемента {0} равен {1}", element, i);
```

Цикл *for* не рекомендуется использовать для задачи поиска, т.к. такое решение не будет являться оптимальным: цикл for проверит каждый элемент массива, даже если искомый элемент будет найден в начале массива.

4.4.2 Поиск индекса последнего вхождения элемента в массив

Необходимо найти индекс последнего вхождения элемента *element* в массив *a*, состоящего из *n* элементов. Алгоритм поиска последнего вхождения элемента в массив аналогичен алгоритму поиска первого вхождения, с той лишь разницей, что цикл следует организовать с конца массива:

```
    int element = int.Parse(Console.ReadLine());
    int i = n-1;
    while (i > 0 && a[i] != element)
        i--;
    Console.WriteLine("Индекс последнего вхождения элемента {0} равен {1}", element, i);
```

4.4.3 Сортировка массивов

Сортировка пузырьком (Bubble sort)

Проходим по массиву слева направо, начиная с первого элемента.

Сравниваем текущий элемент со следующим: если следующий элемент меньше, то меняем их местами.

После первой итерации самый большой элемент окажется последним – на правильном месте. Следовательно, в следующей итерации его учитывать не нужно.

Сортировка выбором (Selection sort)

Идея алгоритма состоит в поиске минимального элемента после текущего.

Проходим по массиву слева направо начиная с первого элемента.

Принимаем *i*-ый элемент массива за минимальный.

Сравниваем его с остальными элементами и среди них находим минимальный.

Меняем местами минимальный и *i*-ый элементы.

Отметим, что после *i*-ой итерации на месте будут стоять первые *i* элементов.

Сортировка вставками (Insertion sort)

Идея алгоритма заключается в том, что на каждой итерации мы передвигаем элемент, начиная со второго, в начало массива до тех пор, пока он больше предыдущего.

Сортировка Шелла (Shell sort)

Данная сортировка похожа на «пузырек», однако на первом шаге происходит смена не соседних элементов, а тех, которые отстоят друг от друга на половину длины массива. На втором – через четверть, на третьем – через 1/8 и так далее. На последней итерации меняются соседние элементы и все окажется отсортированным.

Сортировка перемешиванием (Shaker sort)

Заметим, что сортировка пузырьком работает медленно на тестах, в которых маленькие элементы стоят в конце (их еще называют «черепахами»). Такой элемент на каждом шаге алгоритма будет

сдвигаться всего на одну позицию влево. Идея данной сортировки заключается в том, чтобы идти не только слева направо, но и справа налево. Следует поддерживать два указателя begin и end, обозначающих, какой отрезок массива еще не отсортирован. На очередной итерации при достижении end вычитается из него единица и продолжается движение справа налево, аналогично, при достижении begin прибавляется единица и продолжается движение слева направо.

Гномья сортировка (Gnome sort)

Алгоритм похож на сортировку вставками. Идея алгоритма следующая: поддерживаем указатель на текущий элемент, если он больше предыдущего или он первый – смещаем указатель на позицию вправо, иначе меняем текущий и предыдущий элементы местами и смещаемся влево.

4.5 Лабораторная работа 4. Одномерные массивы

В ранее созданное решение добавить новый проект для лабораторной работы 4.

Написать консольную программу, содержащую лабораторную работу 3 с дополнениями.

К программе добавить пункт меню «Сортировка».

Написать программу, сортирующую массив из n элементов двумя видами сортировки. Сравнить время выполнения сортировок.

В программе должны быть реализованы методы заполнения одномерного массива случайными значениями, вывода одномерного массива на экран, методы сортировок и другие необходимые методы.

№	Варианты задания
1	Сортировка пузырьком. Сортировка перемешиванием.
2	Сортировка пузырьком. Сортировка вставками.
3	Сортировка пузырьком. Сортировка выбором.

4	Сортировка пузырьком. Сортировка Шелла.
5	Гномья сортировка. Сортировка вставками.
6	Гномья сортировка. Сортировка Шелла.
7	Гномья сортировка. Сортировка перемешиванием.
8	Гномья сортировка. Сортировка выбором.
9	Сортировка Шелла. Сортировка вставками.
10	Сортировка Шелла. Сортировка выбором.
11	Сортировка Шелла. Сортировка перемешиванием.

4.6 Вопросы для самоконтроля

1) Что такое массив?
2) К какому типу относятся массивы?
3) Синтаксис объявления одномерного массива.
4) Для чего нужен оператор *new*?
5) Значения по умолчанию различных типов.
6) Что означает значение null?
7) Как получить доступ к элементам массива?
8) Способы инициализации массива.
9) Что представляет собой класс Random?
10) Какой метод позволяет генерировать случайные целые числа? Параметры метода.
11) Какой метод позволяет генерировать случайные целые числа с плавающей запятой от 0,0 до 1,0?
12) В чем отличие метода Sample от метода NextDouble?
13) Для чего предназначен класс Array?
14) Какой метод получает общее число элементов во всех измерениях массива?
15) Какой метод выполняет поиск указанного объекта внутри всего одномерного массива и возвращает индекс его первого вхождения?
16) С помощью какого цикла следует выполнять поиск первого/последнего вхождения элемента в массив?

17) *Какое свойство возвращает число элементов в массиве? Можно ли изменять число элементов массива в процессе выполнения программы? Приведите пример.

18) *Какие методы возвращают максимальное, минимальное значение, содержащееся в массиве? Приведите пример.

19) *Для чего предназначен класс Stopwatch.

20) Дан одномерный массив размером n и два числа a и b (a<b). Определить, сколько элементов массива лежит между a и b.

21) Определить значение и номер последнего отрицательного элемента массива размером n.

22) Найти сумму квадратов элементов, расположенных до первого отрицательного элемента массива.

23) Дан одномерный массив размером 10. Найти сумму элементов, расположенных до максимального элемента массива.

24) В одномерном массиве все элементы, расположенные после максимального, заменить средним значением элементов массива.

5 МНОГОМЕРНЫЕ МАССИВЫ

Одномерные массивы позволяют задавать такие математические структуры, как векторы, двумерные – матрицы, трехмерные – кубы данных, массивы большей размерности – многомерные кубы данных.

Размерность массива – это характеристика типа. Синтаксически размерность массива указывается за счет использования запятых.

Синтаксис объявления и создания двумерного массива аналогичен синтаксису для объявления и создания одномерного массива:

```
тип_элементов[,] имя_массива = new
тип_элементов[число_строк, число_столбцов];
```

Что касается объявления двумерных массивов, то все, что сказано для одномерных массивов, справедливо и для многомерных.

Объявление и инициализация многомерных массивов показана в примере 5.1.

Пример 5.1. Объявление и инициализация многомерных массивов.

```
// Объявление двумерного массива
int[,] multiDimensionalArray1 = new int[2, 3];

// Объявление и инициализация двумерного массива
int[,] multiDimensionalArray2 = {
{ 1, 2, 3 },
{ 4, 5, 6 }
};
```

```
// Объявление ступенчатого массива
int[][] jaggedArray = new int[3][];
jaggedArray [0] = new int[] {5, 7, 9, 11};
jaggedArray [1] = new int[] { 2, 8 };
jaggedArray [2] = new int[] { 6, 12, 4 };
```

5.1 Методы ввода и вывода двумерных массивов

Явная инициализация с использованием многомерных константных массивов, как в примере 5.1, возможна, но применяется редко из-за громоздкости такой структуры. Напишем метод, реализующий заполнение двумерного массива случайными значениями (пример 5.2).

Пример 5.2. Заполнить двумерный массив случайными значениями.

```
    private static void InputArray(int[,] arr, int n,
int m)
    {
        Random rnd = new Random();
        for (int i = 0; i < n; i++)
            for (int j = 0; j < m; j++)
                arr[i, j] = rnd.Next(-10, 10);
    }
```

Метод вывода двумерного массива на консоль показан в примере 5.3.

Пример 5.3. Вывести на консоль двумерный массив

```
    private static void PrintArray(int[,] arr)
    {
        for (int i = 0; i < arr.GetLength(0); i++)
```

```
        {
            for (int j = 0; j < arr.GetLength(1); j++)
                Console.Write(arr[i, j] + "{0}",
    j < arr.GetLength(1) - 1 ? " " : "");
            Console.WriteLine();
        }
    }
```

5.2 Лабораторная работа 5. Работа с двумерными массивами

В ранее созданное решение добавить новый проект для лабораторной работы 5.

Написать консольную программу, содержащую лабораторную работу 4 с дополнениями.

К программе добавить пункт меню «Игра» и реализовать игру, соответствующую варианту.

В программе должны быть реализованы метод заполнения массива и метод вывода массива, отрисовка игрового поля в консоли. Решение задачи должно быть реализовано в методах.

Дополнительное задание. Добавить пункты меню «Загрузить» и «Сохранить». Написать метод загрузки поля из текстового файла. Написать метод сохранения текущего состояния поля в текстовый файл.

1 Создать игру «Крестики-нолики» 3x3. Игроки по очереди ставят фигуры на свободные клетки поля. Первый, выстроивший в ряд 3 своих фигуры по вертикали, горизонтали или диагонали, выигрывает. Первый ход делает игрок, ставящий крестики. Написать алгоритм проверки окончания игры и вывести сообщение о результате. Номер клетки хода пользователи вводят по очереди в консоли в формате номер строки, номер столбца.

2 Создать игру «Крестики-нолики» 4х4. Игроки по очереди ставят фигуры на свободные клетки поля. Первый, выстроивший в ряд 3 своих фигуры по вертикали, горизонтали или диагонали, выигрывает. Первый ход делает игрок, ставящий крестики. Написать алгоритм проверки окончания игры и вывести сообщение о результате. Номер клетки хода пользователи вводят по очереди в консоли в формате номер строки, номер столбца.

3 Создать игру «Морской бой» (для одного игрока). Игровое поле – квадрат 10×10. Клетки поля нумеруются от верхнего левого угла цифрами. На поле размещаются:

- 1 корабль – ряд из 4 клеток («четырёхпалубный»; линкор; цифра 4)
- 2 корабля – ряд из 3 клеток («трёхпалубные»; крейсера; цифра 3)
- 3 корабля – ряд из 2 клеток («двухпалубные»; эсминцы; цифра 2)
- 4 корабля – 1 клетка («однопалубные»; торпедные катера; цифра 1).

При размещении корабли не могут касаться друг друга сторонами и углами.

Написать алгоритм расстановки кораблей на поле. В начале игры пользователь видит пустое игровое поле. После каждого хода обновлять значения клеток поля (0 – промах, 4 – попал в линкор, 3 – попал в крейсер, 2 – попал в эсминца, 1 – попал в торпедный катер) и выводить сообщение о том, какой корабль подбит.

Номер клетки хода пользователь вводит в консоли в формате номер строки, номер столбца.

Написать проверку окончания игры.

4 Создать игру «Сапёр». Игровое поле – квадрат 10×10. Заполнить случайным образом поле N минами (идентификатор мины – цифра 9). Заполнить клетки, не занятые миной, цифрами от 0 до 8 – количество мин, которые расположены в смежных клетках.

В начале игры пользователь видит пустое игровое поле. После каждого хода обновлять значения клеток поля и выводить сообщение, если пользователь попал на клетку с миной. Номер клетки хода пользователь вводит в консоль в формате номер строки, номер столбца.

5 Создать игру «Реверси». Игровое поле – квадрат 8×8. Клетки доски нумеруются от верхнего левого угла цифрами. Один из игроков играет белыми (цифра 0), другой – чёрными (цифра 1). Делая ход, игрок ставит фишку на клетку доски «своим» цветом вверх.

В начале игры в центр доски выставляются 4 фишки: чёрные на 4, 4 и 5, 5, белые на 4, 5 и 5, 4.

	1	2	3	4	5	6	7	8
1								
2								
3								
4				1	0			
5				0	1			
6								
7								
8								

Первый ход делают чёрные. Далее игроки ходят по очереди.

Если после совершения хода фишки соперника находятся в «закрытом» ряду, то они переворачиваются на другую сторону (меняют цвет) и переходят к ходившему игроку. Ряд считается закрытым, если одна или несколько фишек оказываются между фишками противника по горизонтали или по вертикали.

Если в результате одного хода закрывается одновременно более одного ряда фишек противника, то переворачиваются все фишки, оказавшиеся на всех «закрытых» рядах.

Игра прекращается, когда на доску выставлены все фишки. По окончании игры проводится подсчёт фишек каждого цвета, и игрок, чьих фишек на доске выставлено больше, объявляется победителем. В случае равенства количества фишек засчитывается ничья.

6 Создать двумерный массив размером 8x8, заполненный цифрами 0, представляющий собой поле для игры в шахматы. Случайным образом поставить на поле фигуру цифрой 1. В качестве фигуры может быть «Ладья», «Конь» или «Слон» (должна быть предусмотрена возможность выбора). Напишите программу, которая отмечает все поля, которые бьёт фигура цифрой 2.

7 Создать двумерный массив размером 8x8, заполненный цифрами 0, представляющий собой поле для игры в шахматы. Расставить 8 ферзей так, чтобы ни один из них не находился под боем другого. Предполагается, что ферзь бьёт клетки, находящиеся по вертикалям, горизонталям и обеим диагоналям.

5.3 Вопросы для самоконтроля

1) Как объявить и инициализировать двумерный массив?
2) Как объявить и инициализировать ступенчатый массив?
3) В чем отличие двумерного массива от ступенчатого массива?
4) Какого типа могут быть элементы двумерного массива?
5) Дан двумерный массив. Посчитать сумму элементов главной диагонали.
6) Дан двумерный массив. Посчитать сумму элементов побочной диагонали.
7) Дан двумерный массив. Посчитать среднее арифметическое элементов, расположенных над главной диагональю.
8) Дан двумерный массив. Посчитать среднее арифметическое элементов, расположенных под побочной диагональю.
9) Дан двумерный массив. Найти среднее арифметическое элементов массива.

10) Дан двумерный массив. Найти сумму положительных элементов массива.

11) Дан двумерный массив и два числа: i и j. Поменяйте в массиве столбцы с номерами i и j и выведите результат.

12) Дан двумерный массив. Поменяйте в нем первую и последнюю строку. Полученный массив выведите на экран.

6 РАБОТА СО СТРОКАМИ

6.1 Символьный тип данных. Структура Char

В C# есть символьный тип данных *char* (*System.Char*) и использующий двухбайтную кодировку Unicode представления символов. Для этого типа в языке определены символьные константы – символьные литералы. Константу можно задавать:

- символом, заключенным в одинарные кавычки;
- escape-последовательностью;
- Unicode-последовательностью, задающей Unicode код символа.

В примере 6.1 приведено объявление символьных переменных и работа с ними.

Пример 6.1. Работа символьным типом данных

```
char ch1 = 'A';
char ch2 = '\x5A';
char ch3 = '\u0058';
// Преобразование char в int
int code = ch1;
ch2 = (char)(code + 1);
Console.WriteLine("ch1= {0}, ch2= {1}, ch2 = {2},
code = {3}",
ch1, ch2, ch3, code);
```

Результат работы данного кода:

ch1= A, ch2= B, ch2 = X, code = 65

Основные методы и свойства для работы с типом Char приведены в табл. 6.1.

Таблица 6.1. **Статические методы и свойства типа Char**

Метод	Описание
GetNumericValue	Возвращает численное значение символа, если он является цифрой, и (-1) в противном случае
GetUnicodeCategory	Все символы разделены на категории. Метод возвращает Unicode категорию символа.
IsControl	Возвращает true, если символ является управляющим
IsDigit	Возвращает true, если символ является десятичной цифрой
IsLetter	Возвращает true, если символ является буквой
IsLetterOrDigit	Возвращает true, если символ является буквой или цифрой
IsLower	Возвращает true, если символ задан в нижнем регистре
IsNumber	Возвращает true, если символ является числом (десятичной или шестнадцатиричной цифрой)
IsPunctuation	Возвращает true, если символ является знаком препинания
IsSeparator	Возвращает true, если символ является разделителем
IsSurrogate	Некоторые символы Unicode с кодом в интервале [0x10000, 0x10FFF] представляются двумя 16-битными "суррогатными" символами. Метод возвращает true, если символ является суррогатным
IsUpper	Возвращает true, если символ задан в верхнем регистре
IsWhiteSpace	Возвращает true, если символ является "белым пробелом". К белым пробелам, помимо пробела, относятся и другие символы, например, символ конца строки и символ перевода каретки

Окончание таблицы 6.1

Метод	Описание
Parse	Преобразует строку в символ. Естественно, строка должна состоять из одного символа, иначе возникнет ошибка
ToLower	Приводит символ к нижнему регистру
ToUpper	Приводит символ к верхнему регистру
MaxValue, MinValue	Свойства, возвращающие символы с максимальным и минимальным кодом. Возвращаемые символы не имеют видимого образа

6.2 Строковый тип данных. Класс String

Основным типом при работе со строками является тип *string*, задающий строки переменной длины. Класс *String* в языке C# относится к ссылочным типам. Внутри программы текст хранится в виде упорядоченной коллекции объектов Char только для чтения.

Объекты класса String объявляются как все прочие объекты простых типов – с явной или отложенной инициализацией, с явным или неявным вызовом конструктора класса. Чаще всего, при объявлении строковой переменной конструктор явно не вызывается, а инициализация задается строковой константой. Но у класса String достаточно много конструкторов. Они позволяют сконструировать строку из:

- символа, повторенного заданное число раз;
- массива символов char[];
- части массива символов.

Строки можно объявлять и инициализировать различными способами, как показано в следующем примере:

Пример 6.2. Объявление и инициализация строк.

```
string message1;
string message2 = null;
```

```
string greeting = "Hello World!";

char[] letters = { 'A', 'B', 'C' };
string alphabet = new string(letters);
```

Обратите внимание, что оператор *new* не используется для создания объекта строки, за исключением случаев инициализации строки с помощью массива символов.

Операции над строками

Над строками определены следующие операции:

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

Поскольку string – это ссылочный тип, то в результате присваивания создается ссылка на константную строку, хранимую в "куче". С одной и той же строковой константой в "куче" может быть связано несколько переменных строкового типа.

Строковые объекты являются неизменяемыми: их нельзя изменить после создания. Все методы класса *String* и операторы C#, направленные на работу со строками, возвращают результаты в новый строковый объект, а не изменяют их. Когда содержимое строк *s1* и *s2* объединяется для формирования одной строки, две исходные строки не изменяются, как показано примере 6.2. Оператор += создает новую строку, которая содержит объединенное содержимое. Этот новый объект присваивается переменной *s1*, а исходный объект, который был присвоен *s1*, освобождается для сборки мусора, так как ни одна переменная не ссылается на него.

Пример 6.3. Объединение строк.

```
string s1 = "Строки ";
string s2 = "не изменяются.";
```

```
s1 += s2;
Console.WriteLine(s1);
```

В отличие от других ссылочных типов, операции, проверяющие эквивалентность, сравнивают значения строк, а не ссылки. Эти операции выполняются как над значимыми типами.

Бинарная операция " + " сцепляет две строки, приписывая вторую строку к хвосту первой.

Возможность взятия индекса при работе со строками отражает тот приятный факт, что строку можно рассматривать как массив и получать без труда каждый ее символ. Каждый символ строки имеет тип char, доступный только для чтения, но не для записи.

Основные методы и свойства для работы с типом String приведены в табл. 6.2.

Таблица 6.2. **Статические методы и свойства класса String**

Метод	Описание
Empty	Возвращается пустая строка. Свойство со статусом read only
Compare	Сравнение двух строк. Метод перегружен. Реализации метода позволяют сравнивать как строки, так и подстроки. При этом можно учитывать или не учитывать регистр, особенности национального форматирования дат, чисел и т.д.
CompareOrdinal	Сравнение двух строк. Метод перегружен. Реализации метода позволяют сравнивать как строки, так и подстроки. Сравниваются коды символов
Concat	Конкатенация строк. Метод перегружен, допускает сцепление произвольного числа строк
Copy	Создается копия строки
Format	Выполняет форматирование в соответствии с заданными спецификациями формата. Ниже приведено более полное описание метода

Окончание табл. 6.2

Метод	Описание
Intern, IsIntern	Отыскивается и возвращается ссылка на строку, если таковая уже хранится во внутреннем пуле данных. Если же строки нет, то первый из методов добавляет строку во внутренний пул, второй - возвращает null. Методы применяются обычно тогда, когда строка создается с использованием построителя строк - класса StringBuilder
Join	Конкатенация массива строк в единую строку. При конкатенации между элементами массива вставляются разделители. Операция, заданная методом Join, является обратной к операции, заданной методом Split. Последний является динамическим методом и, используя разделители, осуществляет разделение строки на элементы

6.3 Лабораторная работа 6. Работа со строками

В ранее созданное решение добавить новый проект для лабораторной работы 6. Написать консольную программу, содержащую лабораторную работу 5 с дополнениями. Добавить в программу пункт меню для работы со строками. Предусмотреть работу с тестовыми строками и работу со строками, которые вводит пользователь. Решение должно быть реализовано в методах.

№	Задание	Строки
1	Посчитать количество строчных и прописных букв в строке (отдельно).	Строка 1: Варкалось. Хливкие шорьки Пырялись по наве, И хрюкотали зелюки, Как мюмзики в мове. О бойся Бармаглота, сын! Он так свирлеп и дик, А в глуще рымит исполин -
2	Посчитать количество гласных и согласных букв в строке (отдельно)..	
3	Посчитать количество слов в строке.	

№	Задание	Строки
4	Посчитать количество знаков препинания в строке.	Злопастный Брандашмыг.
5	Посчитать количество букв А(а) в строке.	
6	Посчитать количество цифр в строке.	Строка 1: Вот 1 (иль единица), Очень тонкая, как спица, А вот это цифра 2. Полюбуйся, какова: Выгибает двойка шею, Волочится хвост за нею. А за двойкой – посмотри – Выступает цифра 3. Тройка – третий из значков – Состоит из двух крючков.
7	Даны три строки. Вывести их на экран по порядку увеличения символов в них (без пробелов).	Строка 1: Как уже ветерок весенний Строка 2: Поселился в зеленом ростке. Строка 3: Не успела отнять руки,
8	Даны две строки. У них одинаковые начала. Вывести на экран, сколько первых символов этих строк совпадают.	Строка 1: Быть может, вся Природа – мозаика цветов? Строка 2: Быть может, вся Природа – различность голосов?

6.4 Вопросы для самоконтроля

1) Как называется символьный тип данных?
2) Как объявить символьную переменную?
3) Какой метод типа char возвращает true, если символ задан в нижнем регистре?
4) Какой метод типа char приводит символ к верхнему регистру?
5) К какому типу относится строковый тип данных String?
6) Как объявить и инициализировать строку?
7) Какие операции определены над строками?
8) Можно ли изменить строку после создания?

9) Какой метод приводит символ к нижнему регистру?

10) Какой метод приводит символ к верхнему регистру?

11) *Какой метод показывает, относится ли символ к категории знаков препинания?

12) *Какой метод показывает, относится ли символ к категории букв.?

13) *Какой метод показывает, относится ли символ к категории цифр?

14) Дана строка s. Найти индекс первого вхождения символа в строку. Символ вводит пользователь.

15) Дана строка s. Определить процентное отношение строчных и прописных букв к общему числу символов в нем.

16) Дана строка s. Перевернуть её.

7 ОБЪЕКТНО-ОРИЕНТИРОВАННОЕ ПРОГРАММИРОВАНИЕ

7.1 Принципы объектно-ориентированного программирования

C# обеспечивает полную поддержку объектно-ориентированного программирования (ООП), включая основные принципы: инкапсуляция, наследование, полиморфизм.

Инкапсуляция представляет собой способности языка скрывать излишние детали реализации от пользователя объекта.

Например, для управления первым паровым автомобилем необходимо было знать, как устроен паровой котёл, подбрасывать уголь, следить за температурой воды и т.д. При этом для поворота колёс использовались два рычага, каждый из которых поворачивает одно колесо в отдельности. Сейчас все эти действия скрыты от пользователя и позволяют ему управлять автомобилем, не задумываясь, что в это время происходит с инжектором, дроссельной заслонкой и распредвалом. Сокрытие внутренних процессов от пользователя и является инкапсуляцией.

Полиморфизм – это свойство системы использовать объекты с одинаковым интерфейсом без информации о типе и внутренней структуре объекта.

Например, основные элементы управления автомобилем имеют одну и ту же конструкцию и принцип действия. Водитель точно знает, что для того, чтобы повернуть налево, он должен повернуть руль, независимо от марки своего автомобиля, т.к. внутреннее функционирование машины и интерфейс останется прежним.

Наследование описывает возможность создания новых классов на основе существующих классов, т.е. позволяет расширять поведение базового (или родительского) класса, наследуя основную функциональность в производном подклассе (также именуемом дочерним классом).

Например, автомобиль BMW X6 имеет 3 поколения. У этих поколений есть общие характеристики и функциональность, но каждое следующее поколение отличается от предыдущего некоторыми изменениям. В этом случае мы имеем дело с наследованием.

Классы и объекты – фундаментальные понятия объектно-ориентированного программирования. Класс является основой, для создания объектов. В классе определяются данные и код, который работает с этими данными. Объекты являются экземплярами класса.

Поэтому процесс создания объекта называется созданием экземпляра. Класс создается с помощью ключевого слова *class*. Для иллюстрации мы создадим класс, который содержит информацию о геометрической фигуре круг и назовем его *Circle* (пример 7.1). Имена классов в C# принято писать с большой буквы.

Пример 7.1. Создание класса.

```
class Circle
{
    // Тело класса
}
```

Отметим, что при разработке проектов каждый класс принято описывать в отдельном файле. Для этого нужно в обозревателе решений правой кнопкой мыши нажать на название проекта и в контекстном меню выбрать пункт «Добавить» и «Класс» (рис. 7.1).

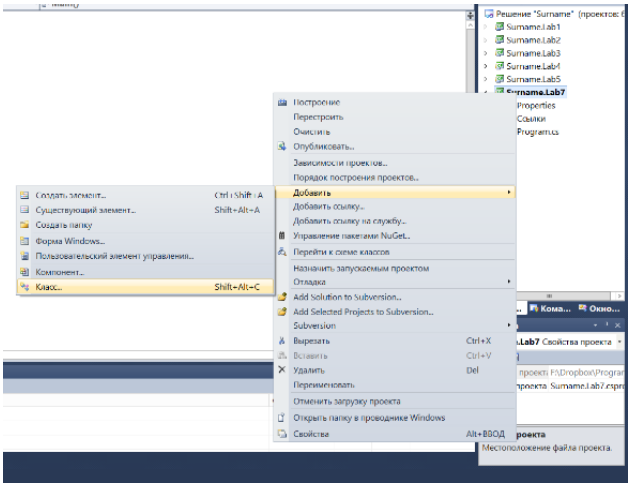

Рис. 7.1. Меню «Добавить класс»

Далее в появившемся окне следует дать название классу и нажать кнопку «Добавить» (рис. 7.2).

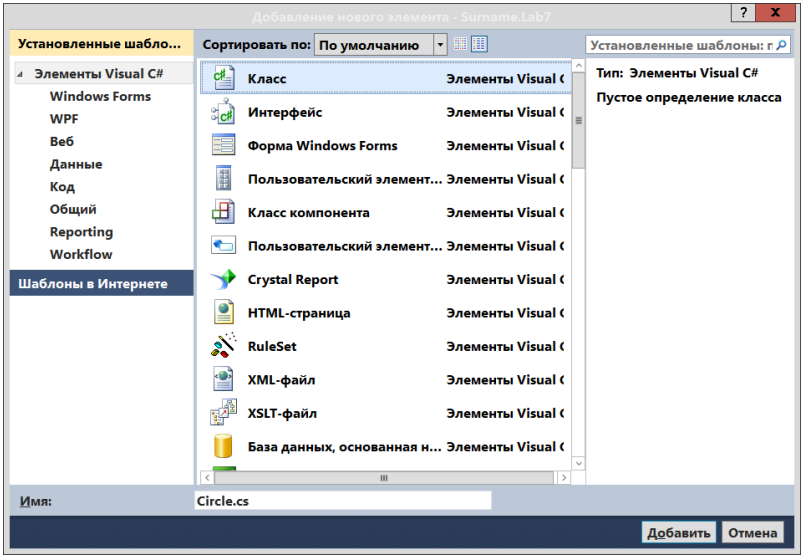

Рис. 7.2. Добавление класса

После этого в проект будет добавлен новый класс (рис. 7.3).

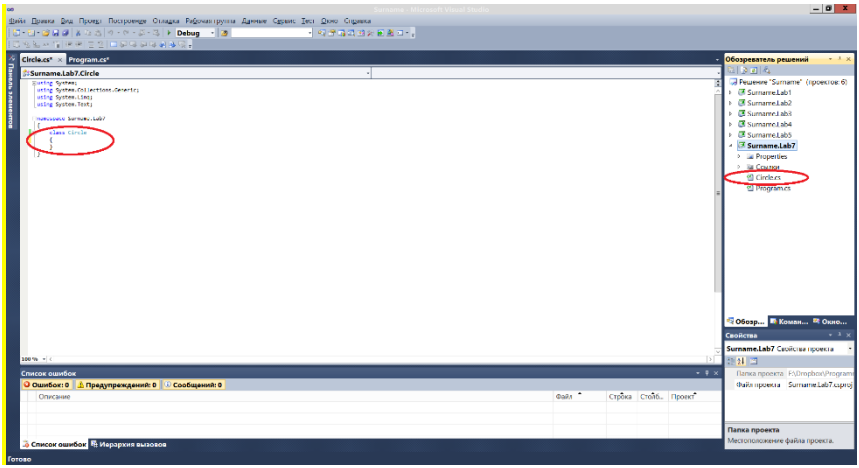

Рис. 7.3. Пример создания класса

7.2 Модификаторы доступа. Уровни доступности

Управление доступом к типам и членам класса в языке C# организуется с помощью четырех модификаторов доступа: *public*, *private*, *protected*, *internal*.

Модификаторы доступа используются для указания одного из следующих объявленных уровней доступности:

– *public* – доступ к типу или члену возможен из любого другого кода в той же сборке или другой сборке, ссылающейся на него. Общий доступ является уровнем доступа с максимальными правами. Ограничений доступа к общим членам не существует.

– *private* – доступ к типу или члену возможен только из кода в том же классе или структуре. Следовательно, методы из других классов не имеют доступа к закрытому члену данного класса. Если ни один из спецификаторов доступа не указан, член класса считается закрытым для своего класса по умолчанию. Поэтому при создании

закрытых членов класса спецификатор private указывать для них необязательно.

– *protected* – доступ к типу или члену возможен только из кода в том же классе либо в классе, производном от этого класса.

– *internal* - доступ к типу или члену возможен из любого кода в той же сборке, но не из другой сборки.

– *protected internal* – доступ к типу или члену возможен из любого кода в той сборке, где он был объявлен, или из производного класса в другой сборке.

– *private protected* – доступ к типу или члену возможен только из его объявляющей сборки из кода в том же классе либо в типе, производном от этого класса.

7.3 Члены класса

Каждый класс состоит из различных *членов класса*, которые содержат свойства, описывающие данные класса, методы, задающие поведение класса, и события, обеспечивающие связь между различными классами и объектами.

Членами класса являются:

– поля;
– конструкторы;
– свойства;
– методы;
– события.

7.3.1 Поля

Синтаксически поля класса являются обычными переменными, т.е. все переменные, объявленные на уровне класса, являются полями класса. Поля класса следует объявлять закрытыми, т.е. с модификатором доступа *private*, поскольку иначе любой метод в любом месте

программы получает неограниченные доступ к внутренним данным объекта. При этом, имена полей рекомендуется начинать с нижнего подчеркивания прописными буквами.

Добавим в класс *Circle* поле радиус и поля, описывающие координату центра круга (пример 7.2).

Пример 7.2. Класс с полями.

```
class Circle
{
    // Радиус
    private double _radius;
    // Координата центра x
    private double _x;
    // Координата центра y
    private double _y;
}
```

7.3.2 Конструкторы

Каждый раз при создании экземпляра класса вызывается конструктор. Конструктор служит для инициализации полей экземпляра класса при его создании. Класс может иметь несколько конструкторов, принимающих различные аргументы. Если класс не содержит конструктор, автоматически создается конструктор по умолчанию, который инициализирует поля класса значениями по умолчанию.

Конструктор имеет следующий синтаксис:

```
модификатор_доступа Имя_класса(список параметров)
{
    // Тело конструктора
}
```

Отметим, что конструктор должен называться так же, как и класс.

Добавим к классу *Circle* конструктор с параметрами (пример 7.3) и конструктор без параметров (пример 7.4).

Пример 7.3. Конструктор с параметрами.

```
public Circle(double x, double y, double radius)
{
    _x = x;
    _y = y;
    Radius = radius;
}
```

Пример 7.4. Конструктор без параметров.

```
public Circle()
{
    _x = 9.5;
    _y = -9.5;
    Radius = 6;
}
```

В показанных примерах *Radius* является свойством, понятие которого рассмотрим далее.

7.3.3 Свойства

Свойство – это член класса, позволяющий читать, записывать или вычислять значение закрытого поля. Свойства сочетают в себе возможности полей и методов. Для пользователя, использующего объект, свойство представляется как поле. При реализации свойство представляется как блок из двух методов: *get* и/или *set*. Блок кода для метода *get* выполняется только при считывании свойства, а блок коды для метода *set* – при записи нового значения. Добавим

в класс *Circle* свойство для чтения и записи радиуса круга (пример 7.5).

Пример 7.5. Свойство для чтения и записи

```
public double Radius
{
    get
    {
        return _radius;
    }
    set
    {
        if (value > 0)
            _radius = value;
        else
            Console.WriteLine("Ошибка");
    }
}
```

7.3.4 Методы

Метод – это блок кода, содержащий набор инструкций. В хорошей программе один метод выполняет только одну задачу. Методы объявляются в классе путем указания уровня доступа, необязательных модификаторов, возвращаемого значения, имени метода и списка параметров этого метода. Все вместе эти элементы образуют сигнатуру метода. Формат записи метода следующий:

```
    модификатор_доступа тип_возврата
Имя_метода(список_параметров)
    {
        // Тело метода
    }
```

Модификатор доступа является необязательным, и, если он не указан, подразумевается, что метод закрыт (*private*) в рамках класса, где он определен.

С помощью элемента *тип_возврата* указывается тип значения, возвращаемого методом. Это может быть любой допустимый тип, включая типы классов, создаваемые программистом. Если метод не возвращает никакого значения, необходимо указать тип *void*.

Элемент *список_параметров* представляет собой последовательность пар (состоящих из типа данных и идентификатора), разделенных запятыми. Параметры – это переменные, которые получают значения аргументов, передаваемых методу при вызове. Если метод не имеет параметров, то список параметров остается пустым.

В теле метода записываются инструкции (блок операторов), которые необходимо выполнить.

Добавим в класс *Circle* метод, вычисляющий площадь круга (пример 7.6) – метод с параметрами, и метод, определяющий, попадает ли точка с заданными координатами в круг (пример 7.7) – метод без параметров.

Пример 7.6. Метод без параметров.

```
public double CalculateSquare()
{
    double square = Math.PI * _radius * _radius;
    return square;
}
```

Пример 7.7. Метод с параметрами.

```
public bool Inside(double x, double y)
{
    // Если inside=true - точка внутри круга
    bool inside = false;
    if (Math.Pow(_x - _radius, 2) + Math.Pow(_y -
_radius, 2) < Math.Pow(_radius, 2))
```

```
            inside = true;
        return inside;
    }
```

7.4 Работа с экземпляром класса

Для создания экземпляра класса *Circle* используется оператор *new*:

```
Circle circle1 = new Circle();
```

В левой части данной инструкции объявляется переменная *circle1* типа *Circle*. При помощи оператора *new* выделяется память для экземпляра класса и вызывается конструктор без параметров.

Создадим еще один экземпляр класса *Circle*, используя конструктор с параметрами:

```
double x = 0;
double y = 0;
double r = 10;
Circle circle2 = new Circle(x, y, r);
```

Для доступа к членам класса используется оператор "точка" (.). У экземпляров класса *Circle* все поля являются закрытыми, поэтому получить к ним доступ нельзя, а для изменения и чтения доступно свойство *Radius*. В примере 7.8 показано, как изменить значение радиуса для экземпляра *circle1*. В данном случае у свойства *Radius* выполнится блок метода *set*.

Пример 7.8. Доступ к свойству для записи.

```
Console.WriteLine("Введите радиус: ");
circle1.Radius = double.Parse(Console.ReadLine());
```

Аналогично реализуется доступ к методам. Рассчитаем и выведем на консоль площадь круга (пример 7.9).

Пример 7.9. Доступ к методу.

```
    double square = circle2.CalculateSquare();
    Console.WriteLine("Площадь круга равна {0}",
square);
```

В примерах 7.10 и 7.11 приведен код класса *Circle.cs* и код класса *Program.cs*.

7.5 XML-документация

XML-документация – это такие специальные теги XML, которые содержатся в комментариях и описывают свойства или методы в конкретном файле.

Данный вид документации позволяет стандартизировать комментарии проектах на C#, отображать информацию о документированных методах и параметрах так же, как и для методов, встроенных во фреймворк, генерировать XML файл, который будет содержать все комментарии в удобном формате.

Для того, чтобы вводить теги нужно сначала поставить тройной слеш «///». Теги для документирования приведены в табл. 7.1.

Пример документирования кода приведен в классах *Circle.cs* и *Program.cs*.

Таблица 7.1. **XML-теги**

Тег	Описание
<c>	Помечает текст в строке как код
<code>	Помечает множество строк как код
<example>	Помечает пример кода
<exception>	Документирует класс исключения (синтаксис проверяется компилятором)

Окончание табл. 7.1

Тег	Описание
<include>	Включает комментарии из другого файла документации (синтаксис проверяется компилятором)
<list>	Вставляет список в документацию
<param>	Помечает параметр метода (синтаксис проверяется компилятором)
<paramref>	Указывает, что слово является параметром метода (синтаксис проверяется компилятором)
<permission>	Документирует доступ к члену (синтаксис проверяется компилятором)
<remarks>	Добавляет описание члена
<returns>	Документирует возвращаемое методом значение
<see>	Представляет перекрестную ссылку на другой параметр (синтаксис проверяется компилятором)
<seealso>	Представляет раздел “see also” («смотреть также») в описании (синтаксис проверяется компилятором)
<summary>	Представляет краткий итог о типе или члене
<value>	Описывает свойство

Чтобы сохранить XML-файл с документацией необходимо в свойствах проекта во вкладке «Построение» включить сохранение файла (рис. 7.4).

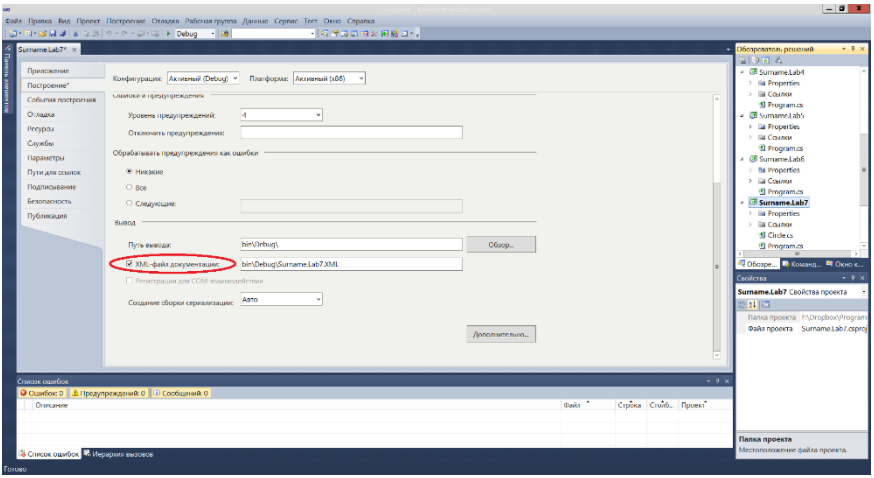

Рис. 7.4. Создание XML-файла

Пример 7.10. Класс Circle.cs.

```
using System;
namespace Surname.Lab7
{
    class Circle
    {
        // Радиус
        private double _radius;
        // Координата центра x
        private double _x;
        // Координата центра y
        private double _y;

        /// <summary>
        /// Конструктор с параметрами.
        /// </summary>
        /// <param name="x"> Координата центра x.
</param>
```

```
        /// <param name="y"> Координата центра y.
</param>
        /// <param name="radius"> Радиус круга.
</param>
        public Circle(double x, double y, double
radius)
        {
            _x = x;
            _y = y;
            Radius = radius;
        }

        /// <summary>
        /// Конструктор без параметров.
        /// </summary>
        public Circle()
        {
            _x = 9.5;
            _y = -9.5;
            Radius = 6;
        }

        /// <summary>
        /// Радиус.
        /// </summary>
        public double Radius
        {
            get
            {
                return _radius;
            }
            set
```

```
            {
                if (value > 0)
                    _radius = value;
                else
                    Console.WriteLine("Ошибка");
            }
        }

        /// <summary>
        /// Метод, вычисляющий площадь круга.
        /// </summary>
        /// <returns> Площадь. </returns>
        public double CalculateSquare()
        {
            double square = Math.PI * _radius *
_radius;
            return square;
        }

        /// <summary>
        /// Метод, определяющий, попадает ли точка
с заданными координатами в круг.
        /// </summary>
        /// <param name="x"> Координата точки x.
</param>
        /// <param name="y"> Координата точки y.
</param>
        /// <returns> Попала/не попала. </returns>
        public bool Inside(double x, double y)
        {
            // если inside=true - точка внутри
круга
```

```
                bool inside = false;
                if (Math.Pow(_x - _radius, 2) +
Math.Pow(_y - _radius, 2) < Math.Pow(_radius, 2))
                    inside = true;
                return inside;
            }
        }
    }
```

Пример 7.11. Класс Program.cs.

```
using System;
namespace Surname.Lab7
{
    class Program
    {
        static void Main()
        {
            Circle circle1 = new Circle();

            double x = 0;
            double y = 0;
            double r = 10;
            Circle circle2 = new Circle(x, y, r);
            Console.WriteLine("Введите радиус: ");
            circle1.Radius = InputDouble();
            double square = cir-
cle2.CalculateSquare();
            Console.WriteLine("Площадь круга равна
{0}", square);
        }
        /// <summary>
        /// Метод ввода вещественного числа
```

```
            /// </summary>
            /// <returns></returns>
            public static double InputDouble()
            {
                double number;
                while (!dou-
ble.TryParse(Console.ReadLine(), out number))
                {
                    Console.WriteLine("Введите число:
");
                }
                return number;
            }
        }
    }
```

7.6 Лабораторная работа 7. Классы

В ранее созданное решение добавить новый проект для лабораторной работы 7.

Написать программу, содержащую лабораторную работу 6 с дополнениями. Реализовать лабораторную работу в виде классов с XML-документированием.

Должны быть созданы следующие классы:

1) Статический класс, содержащий методы для игры по отгадыванию ответа значения функции. В классе должен быть метод, вычисляющий значение функции.

2) Класс, содержащий методы для работы с массивами. Класс должен содержать поле для количества элементов в массиве и поле для массива, а также методы для работы с массивом в соответствии с лабораторной работой 4. В классе должен быть конструктор без параметров, задающий по умолчанию число элементов в массиве рав-

ным 10, и конструктор с параметрами, в который передается число элементов, вводимых пользователем.

3) Статический класс, содержащий методы для проверки вводимых данных. Класс должен содержать методы, обеспечивающие контроль ввода.

4) Класс, содержащий методы для работы со строками. Класс должен содержать константу, задающую тестовую строку, поле для задания строки и методы работы со строками в соответствии с лабораторной работой 6. В классе должен быть конструктор без параметров, задающий строку по умолчанию равной константе, и конструктор с параметрами, в который передается строка, вводимая пользователем.

5) Класс, содержащий реализацию игры. Класс должен содержать константу, определяющую размер поля, константы, определяющие обозначения элементов полей и методы в соответствии с лабораторной работой 5.

При создании классов следует придерживаться принципа разделения логики работы программы и визуализации данных.

Поля классов должны иметь модификатор доступа private. Для полей, к которым необходимо получение доступа из других классов, необходимо написать свойства.

7.7 Вопросы для самоконтроля

1) Назовите основные принципы ООП.
2) Что такое инкапсуляция?
3) Что такое полиморфизм?
4) Что такое наследование?
5) Что такое класс?
6) В чем отличие класса от объекта?
7) Как добавить новый класс в проект в Visual Studio?
8) Какие модификаторы доступа используются в C#?

9) Какой уровень доступа предоставляет модификатор public?

10) Какой уровень доступа предоставляет модификатор private?

11) Какой уровень доступа предоставляет модификатор protected?

12) Какой уровень доступа предоставляет модификатор internal?

13) Какие члены класса может содержать класс?

14) Поля. Синтаксис.

15) Каким уровнем доступа следует наделять поля класса?

16) Конструкторы. Синтаксис.

17) Для чего предназначен конструктор класса?

18) Может ли конструктор возвращать значение?

19) Свойства. Синтаксис.

20) Когда в свойствах выполняется блок с методом get?

21) Когда в свойствах выполняется блок с методом set?

22) Всегда ли в свойствах определены оба метода (get и set)?

23) Методы. Синтаксис.

24) Какой тип указывает на то, что метод ничего не возвращает?

25) Может ли метод не иметь параметров?

26) В чем отличие конструктора от метода?

27) Как создать экземпляр класса? Создайте экземпляр класса Circle.

28) Как реализуется доступ к членам класса? Приведите пример.

29) Что такое XML-документирование?

30) Как создать XML-файл с документацией?

8 WINDOWS-ПРИЛОЖЕНИЯ

8.1 Создание Windows-приложения

Для создания Windows-приложения следует в меню «Файл» выбрать пункт «Создать проект» (или «Добавить» для добавления проекта в уже существующее решение), после чего откроется окно, показанное на рис. 8.1.

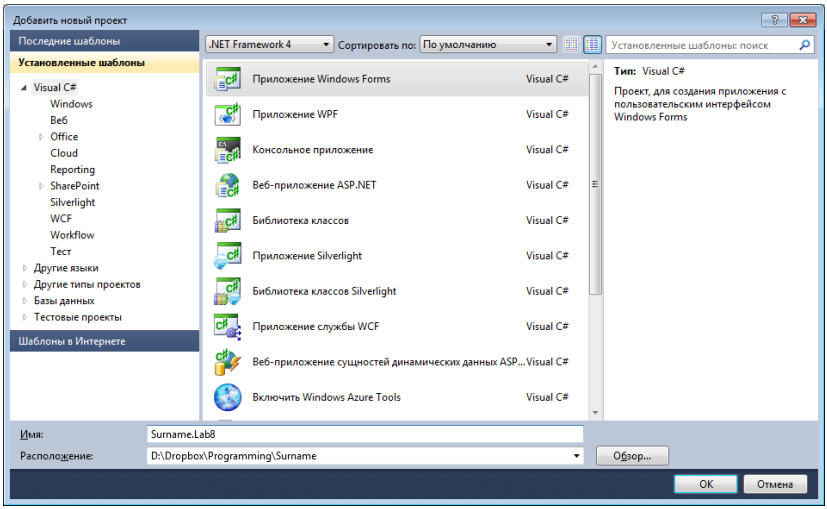

Рис. 8.1. Создание Windows-приложения

Для создания Windows-приложения следует выбрать вид приложения «Приложение Windows Forms» и задать имя проекта в соответствии с рекомендациями. После нажатия на кнопку «OK» откроется окно с главной формой программы, которая по умолчанию называется Form1 (рис. 8.2).

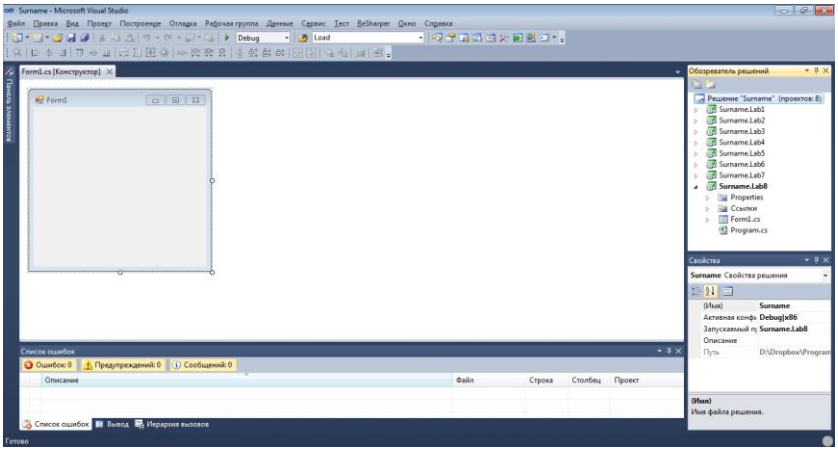

Рис. 8.2. Вид экрана после создания приложения Windows Forms

Перед работой необходимо настроить среду программирования, чтобы были открыты обозреватель решений, список ошибок, окно свойств и панель элементов, как показано на рис. 8.3. Для этого необходимо выбрать соответствующие пункты в меню «Вид». Окно свойств и панель элементов будут необходимы при разработке проектов приложений Windows Forms.

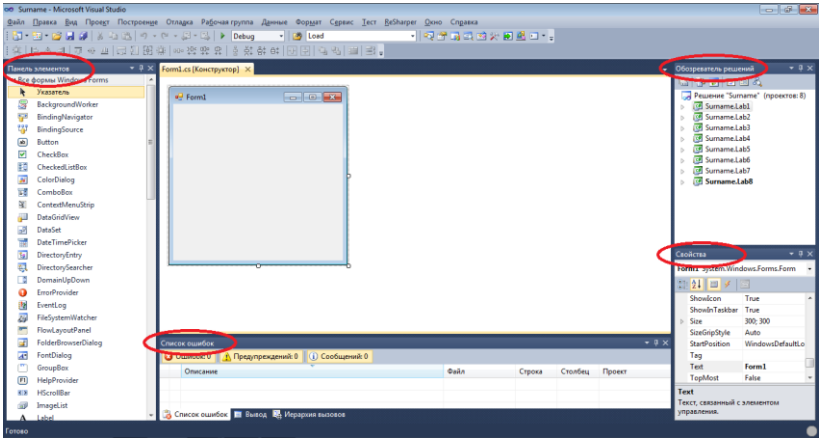

Рис. 8.3. Настройка среды для создания приложения Windows Forms

Создание Windows-приложения включает в себя два этапа: визуальное проектирование и проектирование поведения приложения.

Визуальное проектирование представляет собой добавление компонентов на форму (например, кнопок, полей ввода, меню и др.) и настройку свойств этих компонентов. Свойства компонентов отображаются в окне свойств (рис. 8.4).

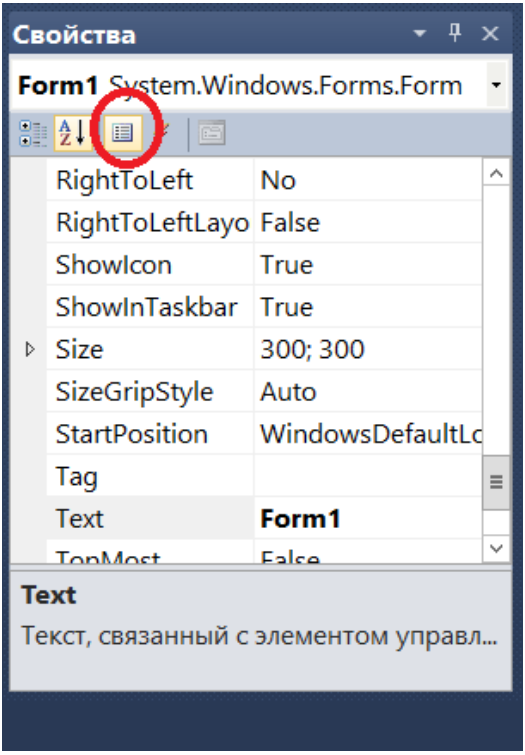

Рис. 8.4. Свойства формы

В табл. 8.1 показаны основные свойства формы. Многие свойства являются универсальными и определены не только для форм, но и для компонентов формы.

Таблица 8.1. **Свойства формы**

Свойство	Описание
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.
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.
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

Окончание таблицы 8.1

Свойство	Описание
MinimizeBox	Указывает, будет ли доступна кнопка минимизации окна.
MaximumSize	Задает максимальный размер формы.
MinimumSize	Задает минимальный размер 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 от пользователя, то можем задать данному свойству значение false.

Проектирование поведения приложения заключается в определении действий, реализующих функциональность программы.

Для взаимодействия с пользователем в Windows Forms используется механизм событий. События в Windows Forms представляют стандартные события на C#, только применяемые к визуальным компонентам и подчиняются тем же правилам, что события в C#. Но создание обработчиков событий в Windows Forms все же имеет некоторые особенности.

Прежде всего в Windows Forms есть некоторый стандартный набор событий, который по большей части имеется у всех визуальных компонентов. Отдельные элементы добавляют свои события, но принципы работы с ними будут похожие. Чтобы посмотреть все события элемента, нам надо выбрать этот элемент в поле графического дизайнера и перейти к вкладке событий на панели форм. Например, события формы показаны на рис. 8.5.

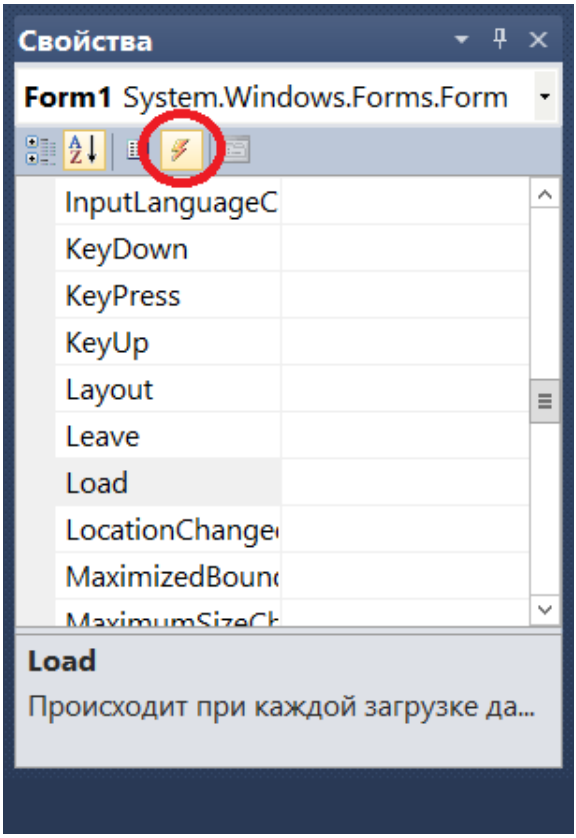

Рис. 8.5. События формы

Чтобы добавить обработчик, можно просто два раза нажать по пустому полю рядом с названием события, и после этого Visual Studio автоматически сгенерирует обработчик события. Например, нажмем для создания обработчика для события Load (рис. 8.6):

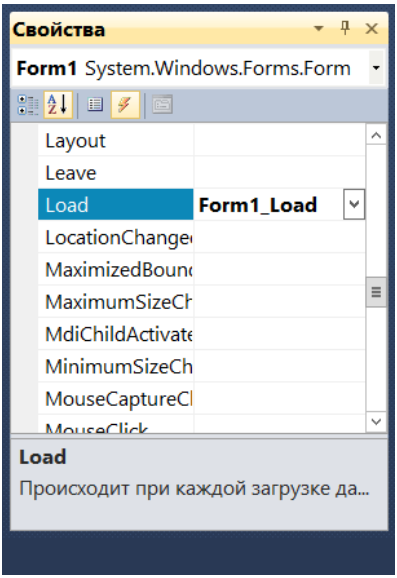

Рис. 8.6. Добавление обработчика события

При этом в код формы автоматически будет добавлен код обработчика события (рис. 8.7).

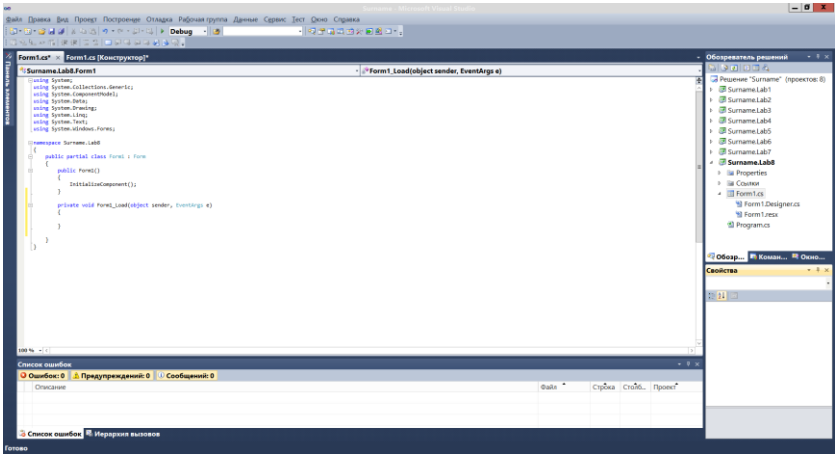

Рис. 8.7. Код обработчика событий Load

8.2 Создание простейшего Windows-приложения

8.2.1 Стандартные элементы управления

Элемент управления Button

Элемент управления Button позволяет пользователю щелкнуть его для выполнения действия. При щелчке кнопки мышью элемент управления выглядит так, как будто его нажимают и отпускают. Когда пользователь нажимает кнопку, вызывается обработчик события Click. Далее необходимо поместить код в обработчик события Click.

Текст, отображаемый на кнопке, содержится в свойстве Text. Внешний вид текста определяется свойствами Font и TextAlign.

Кнопка Button также может отображать изображения, используя свойства Image и ImageList. В табл. 8.2 приведены основные свойства и события элемента управления Button.

Таблица 8.2. **Свойства и события элемента управления Button**

Член класса	Наименование	Описание
Свойства	Name	Возвращает или задает имя элемента управления.
	Enabled	Указывает, включен ли элемент управления.
	Font	Шрифт, используемый для отображения текста на элементе управления.
	Text	Возвращает или задает текст, связанный с этим элементом управления.
	TextAlign	Возвращает или задает способ выравнивания текста на кнопке.
Свойства	Visible	Возвращает или задает значение, указывающее, отображаются ли элемент управления и все его дочерние элементы управления.

Окончание таблицы 8.2

Член класса	Наименование	Описание
	Image	Возвращает или задает изображение, отображаемое на кнопке.
	ImageList	Получает или задает свойство ImageList, содержащее изображение Image, отображенное в кнопке.
События	Click	Происходит при щелчке элемента управления.
	DoubleClick	Происходит при двойном щелчке элемента управления Button.

Добавим на форму 2 кнопки и зададим им в окне свойств следующие свойства (рис. 8.7):

Name = buttonOk, Text = Ok,
Name = buttonCancel, Text = Отмена.

При добавлении элементов управления на форму следует задавать им соответствующее свойство Name.

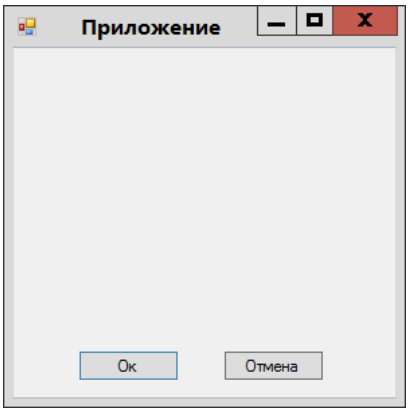

Рис. 8.8. Элемент управления Button

Элемент управления CheckBox

Элемент CheckBox или флажок предназначен для установки одного из двух значений: отмечен или не отмечен. Чтобы отметить флажок, надо установить у его свойства Checked значение true. Кроме свойства Checked у элемента CheckBox имеется свойство CheckState, которое позволяет задать для флажка одно из трех состояний – Checked (отмечен), Indeterminate (флажок не определен - отмечен, но находится в неактивном состоянии) и Unchecked (не отмечен)

Также следует отметить свойство AutoCheck – если оно имеет значение false, то мы не можем изменять состояние флажка. По умолчанию оно имеет значение true.

При изменении состояния флажка он генерирует событие CheckedChanged. Обрабатывая это событие, мы можем получать измененный флажок и производить определенные действия.

В табл. 8.3 приведены основные свойства и события элемента управления Button.

Таблица 8.3. **Свойства и события элемента управления CheckBox**

	Наименование	Описание
Свойство	AutoCheck	Получает или задает значение, указывающее, изменяются ли значения Checked или CheckState, а также внешний вид CheckBox автоматически при щелчке элемента управления CheckBox.
	Checked	Получает или задает значение, определяющее, находится ли CheckBox в выбранном состоянии.
	CheckState	Получает или задает состояние CheckBox.
	Name	Возвращает или задает имя элемента управления.

Окончание табл. 8.3

	Text	Возвращает или задает текст, связанный с этим элементом управления.
	Visible	Возвращает или задает значение, указывающее, отображаются ли элемент управления и все его дочерние элементы управления.
События	CheckedChanged	Происходит при изменении значения свойства Checked.

Добавим на форму 2 элемента CheckBox для выбора операции вычисления (рис. 8.9): сумма и/или произведение. Согласно рекомендациям, свойство Name одного элемента равно checkBoxSum, а другого checkBoxMult.

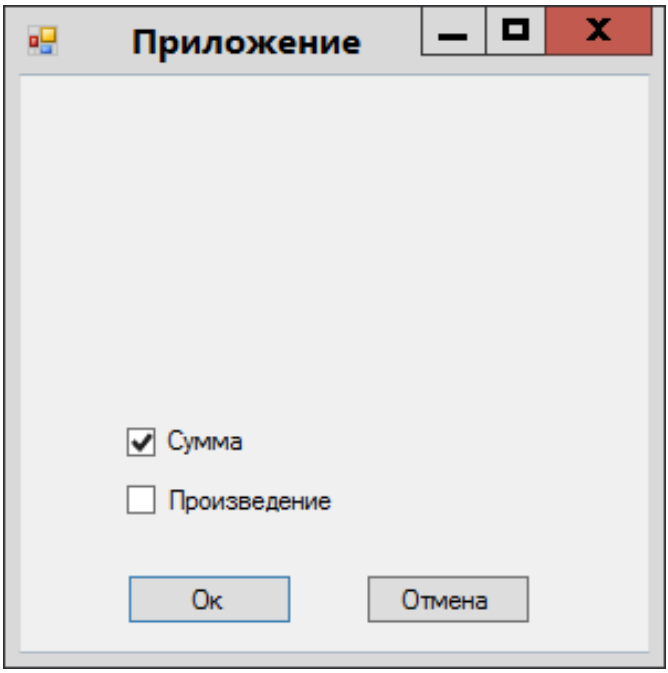

Рис. 8.9. Элемент управления CheckBox

Элемент управления Label

Элемент управления Label используются для отображения текста или изображений, которые не могут быть изменены пользователем. Они используются для идентификации других объектов в форме. На рис. 8.10 добавлены три элемента Label:

Name = labelA, Text = Число 1,
Name = labelB, Text = Число 2,
Name = labeResult, Text = Результат,

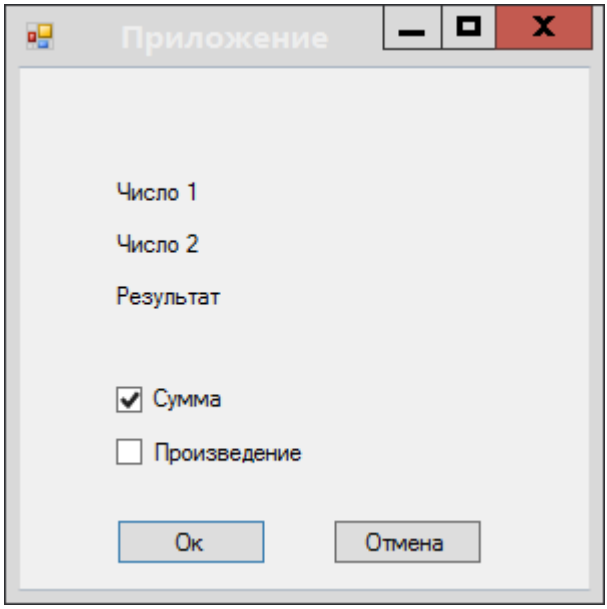

Рис. 8.10. Элемент управления Label

Элемент управления TextBox

Текстовые поля TextBox используются для получения входных данных от пользователя или для отображения текста. Элемент TextBox обычно используется для редактируемого текста, хотя его можно также сделать доступным только для чтения. Для отображе-

ния многострочного форматируемого текста используется элемент RichTextBox.

На рис. 8.11 добавлены поля для ввода данных: textBoxA, textBoxB и textBoxResult.

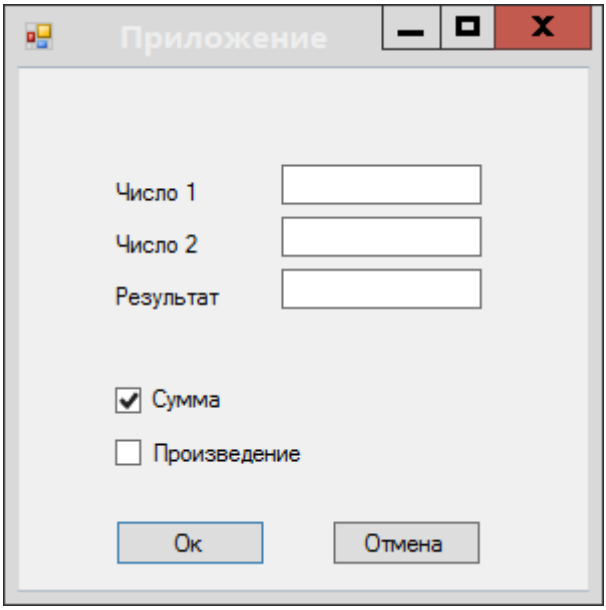

Рис. 8.11. Элемент управления TextBox

8.2.2 Разработка логической составляющей приложения

При разработке приложений следует разделять бизнес-логику приложения от способа ее представления, поэтому добавим в проект класс Calculate для вычислений. Данный класс будет содержать методы для вычисления суммы и произведения (пример 8.1).

Пример 8.1. Класс Calculate.

```
class Calculate
{
```

```
        public static double Sum(double a, double b)
        {
            return a + b;
        }
        public static double Mul(double a, double b)
        {
            return a * b;
        }
    }
```

Добавим обработчик события Click для кнопки Ok, который представлен в примере 8.2.

Пример 8.2. Обработчик события Click для кнопки Ok.

```
    private void buttonOk_Click(object sender, Even-
tArgs e)
    {
        double a = 0;
        double b = 0;
        if (!double.TryParse(textBoxA.Text, out a) ||
!double.TryParse(textBoxB.Text, out b))
            MessageBox.Show("Ошибка ввода!");
        else
        {
            double sum = Calculate.Sum(a, b);
            double mul = Calculate.Mul(a, b);

            if (checkBoxSum.Checked && check-
BoxMult.Checked)
                textBoxResult.Text = sum.ToString() +
"; " + mul.ToString();
            else
```

```
                if (checkBoxSum.Checked)
                    textBoxResult.Text =
sum.ToString();
                else
                    if (checkBoxMult.Checked)
                        textBoxResult.Text =
mul.ToString();
                    else
                        textBoxResult.Text = "";
        }
    }
```

8.3 Лабораторная работа 8. Разработка Windows – приложения

В ранее созданное решение добавить новый проект для лабораторной работы 8.

Создать проект приложения Windows Forms, разработать формы для проекта из лабораторной работы 7. Реализовать работу приложения, добавив и модифицировав классы из предыдущих работ.

Должны быть реализованы следующие формы:

- игра по отгадыванию ответа значения функции;
- работа с одномерными массивами;
- работа со строками;
- игра;
- вывод информации об авторе.

На каждой форме должно быть выведено задание.

Для вывода одномерных массивов следует использовать компонент DataGridView.

Компоненты для вводимых и выводимых данных должны быть подписаны.

При выходе из программы должно запрашиваться подтверждение.

8.4 Вопросы для самоконтроля

1) Как создать приложение Windows Forms?
2) Для чего нужны свойства формы?
3) Каким образом изменить значение свойства?
4) Каким образом получить значение свойства?
5) Что определяют события формы?
6) Какие компоненты формы вы знаете?
7) Какое свойство указывает, включен ли элемент управления?
8) Какое свойство возвращает или задает текст, связанный с этим элементом управления?
9) Какое событие происходит при щелчке элемента управления Button?
10) Какое событие происходит при двойном щелчке элемента управления Button?
11) На форме содержится элемент CheckBox с именем checkBoxVisible. Как программно установить, что его флаг отмечен?
12) Какие компоненты используются для ввода и вывода текста?
13) Какие компоненты используются для отображения меню?
14) Какие виды меню имеются в стандартном наборе компонентов?
15) Какой компонент используется для отображения таблиц?
16) Какой компонент предназначен для рисования?

9 ДОКУМЕНТИРОВАНИЕ

Написать итоговый отчет по всем лабораторным работам, оформив его согласно требованиям нормативной документации.

Пример оформления отчета приведен в приложении А.

СПИСОК ЛИТЕРАТУРЫ

1 Акчурин, Э.А. Программирование на языке C# в Microsoft Visual Studio .NET [Текст] / Э.А. Акчурин. – Самара: ИУНЛ ПГУТИ, 2010. – 128 с.

2 Бхаргава, А. Грокаем алгоритмы [Текст] / А. Бхаргава – СПб.: Питер, 2018. – 288 с.

3 Вирт, Н. Алгоритмы и структуры данных [Текст] / Н. Вирт. – СПб.: ДМК Пресс, 2016. – 272 с.

4 ГОСТ 2.105-95 Единая система конструкторской документации. Общие требования к текстовым документам [Текст]. – Введ. 1996-06-30. – М: Издательство стандартов, 1995. – 31 с.

5 Кнут, Д.Э. Искусство программирования. Том 1. Основные алгоритмы / Д.Э. Кнут. – М.: Вильямс, 2017. – 720 с.

6 Макконелл, Дж. Анализ алгоритмов. Активный обучающий подход [Текст] / Дж. Макконелл. – М.: Техносфера, 2013. – 415 с.

7 Макконнелл, С. Совершенный код [Текст] / С. Макконнелл. – СПб.: БХВ, 2017. –896 с.

8 Мартин, Р. Чистая архитектура. Искусство разработки программного обеспечения [Текст] / Р. Мартин. – СПб.: Питер, 2018. – 352 с.

9 Николаев, Е.И. Объектно-ориентированное программирование [Текст] / Е.И. Николаев. – Ставрополь: СКФУ, 2015. – 225 с.

10 Окулов, С.М. Программирование в алгоритмах [Текст] / С.М. Окулов. – М.: Бином, 2013. – 384 с.

11 Павловская, Т.А. C#. Программирование на языке высокого уровня [Текст] / Т.А. Павловская. – СПб.: Питер, 2014. – 432 с.

12 Портал разработчиков Microsoft Developer Network [Электронный ресурс]. – Режим доступа: https://msdn.microsoft.com/ru-ru.

13 Рефакторинг. Улучшение проекта существующего кода [Текст] / М. Фаулер, К. Бек, Дж. Брант, Д. Робертс. – СПб.: Вильямс, 2017. – 448 с.
14 Рихтер, Дж. CLR via C#. Программирование на платформе Microsoft .NET Framework 4.5 на языке C# [Текст] / Дж. Рихтер. – СПб.: Питер, 2019. – 896 с.
15 Сообщество разработчиков программного обеспечения [Электронный ресурс]. – Режим доступа: https://stackoverflow.com.
16 СТО 02068410-004-2018 Общие требования к учебным текстовым документам [Текст]. – Самара: Самарский университет, 2018. – 36 с.
17 Структуры данных и алгоритмы [Текст] / А.В. Ахо, Дж.Э. Хопкрофт, Дж.Д. Ульман. – М.: Вильямс, 2018. – 400 с.

ПРИЛОЖЕНИЕ А
Пример оформления отчета

МИНИСТЕРСТВО НАУКИ И ВЫСШЕГО ОБРАЗОВАНИЯ
РОССИЙСКОЙ ФЕДЕРАЦИИ

Федеральное государственное автономное
образовательное учреждение высшего образования
«Самарский национальный исследовательский университет
имени академика С.П. Королева»
(Самарский университет)

Институт информатики, математики и электроники
Факультет информатики
Кафедра информационных систем и технологий

ОТЧЁТ
по циклу лабораторных работ
по курсу «Программирование»

Выполнил: Васечкин В.В.,
гр. 6101-090301D

Проверил: уч. степень, уч. звание, должность Иванов И.И.

Самара 2019

СОДЕРЖАНИЕ

1 График выполнения лабораторных работ

Васечкин В.В., гр. 6101-090301D				
№	Вариант	Задание принято к исполнению		Отметка о выполнении
		Дата	Подпись	
1				
2				
3				
4				
5				
6				
7				
8				
9				

2 Лабораторная работа 1. Простейшие программы

2.1 Задание

Написать консольную программу для получения ответа на вопрос и вывода на экран правильного ответа:

1) Пользователь выводит параметры функции.

2) Вывести на экран вопрос «Чему равно значение функции: $f = \sqrt{((\sin a)^2 + (\cos b)^3) / ((\sin a)^3 - (\cos b)^2)}$?

3) Получить ответ пользователя.

4) Вывести на экран правильный ответ «Правильный ответ: <ответ>».

2.2 Разработка программы

В лабораторной работе необходимо написать программу для вычисления значения функции. Список переменных, необходимых для реализации программы, приведен в таблице 1.

Таблица 1 – Переменные программы

Имя	Семантика	Тип	Ограничения
a	Вводимое число	double	–
b	Вводимое число	double	–
result	Значение функции	double	x>=0

Текст программы:

```
static void Main(string[] args)
{
    Con-
sole.WriteLine("F=((Sin^2(A)+Cos^3(B))/Sin^3(A)-
Cos^2(B))^(1/2)");
    Console.Write("Введите переменную a: ");
    double A = double.Parse(Console.ReadLine());
    Console.Write("Введите переменную b: ");
    double B = double.Parse(Console.ReadLine());
    Console.WriteLine("Чему равно значение данной
функции?");
    string P = Console.ReadLine();
    double c = Math.Pow(Math.Sin(A), 2);
    double d = Math.Pow(Math.Cos(B), 3);
    double e = Math.Pow(Math.Sin(A), 3);
    double f = Math.Pow(Math.Cos(B), 2);
    double result = Math.Sqrt((c + d) / (e - f));
    Console.WriteLine("Результат: " + result);
    Console.ReadKey();
}
```

2.3 Демонстрация и тестирование программы

В таблице 2 представлены тесты для проверки работы программы.

Таблица 2 – Набор тестов

№	Входные данные	Результат
1	a=1; b=1	result = 1,68789854355...
2	a=0; b=0	result = не число
3	a=321; b=123	result = 0,81005443052...

Результаты работы программы представлены на рисунках 1-3.

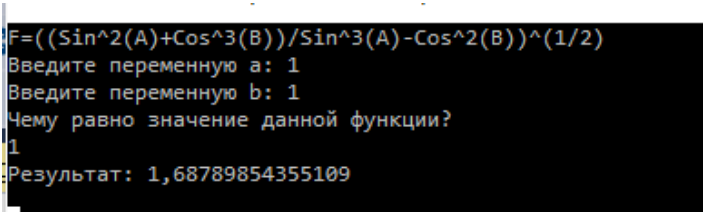

Рисунок 1 – Результат теста № 1

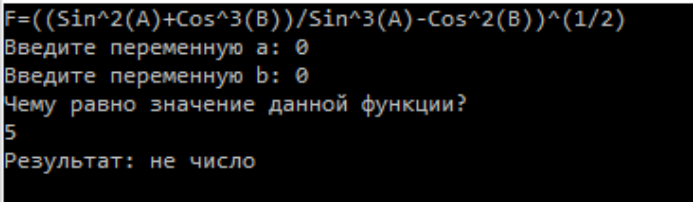

Рисунок 2 – Результат теста № 2

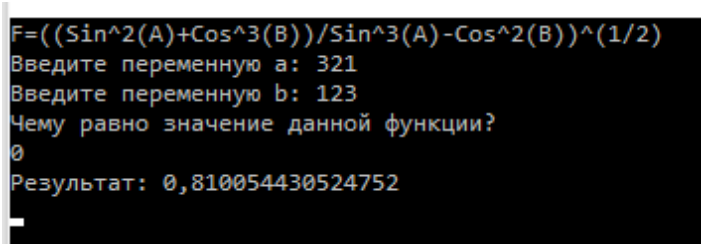

Рисунок 3 – Результат теста № 3

2.4 Результаты

В результате выполнения лабораторной работы изучены основы работы в Visual Studio. Изучены методы классов Console, Math для ввода и вывода данных и работы с математическими функциями. Разработано и протестировано консольное приложение.

3 Лабораторная работа 2. Условные операторы

3.1 Задание

Написать программу, проверяющую ответ пользователя на вопрос из лабораторной работы 1. Ответ округлить. Реализовать меню для выбора действий:

1 – Отгадай ответ

2 – Об авторе (Фамилия И.О., группа)

3 – Задание

4 – Выход

При выборе меню 4 запрашивать подтверждение на выход из программы (при нажатии «д» выйти из программы, при нажатии «н» – остаться в программе).

Реализовать проверку на корректность ввода: сообщать об ошибке, если пользователь вводит буквы вместо цифр.

Предусмотреть корректную работу программы в случае деления на 0, взятия корня из отрицательного числа и т.п.

Разработать тестовые примеры для проверки исключительных ситуаций.

3.2 Разработка программы

В лабораторной работе необходимо написать программу для реализации меню, вычисления функции и проверки на корректность ввода. Список переменных, необходимых для реализации программы, приведен в таблице 3.

Таблица 3 – Переменные программы

Имя	Семантика	Тип	Ограничения
number	Пункт меню	integer	[1;4]
str	Запрос выхода	string	

Текст главной программы:

```
static void Main(string[] args)
{
    while (true)
    {
        int number;
        Console.WriteLine(" 1 - Выполнение лабораторной работы\n 2 - Об авторе\n 3 - Задание\n 4 - Выход ");
        while (!int.TryParse(Console.ReadLine(), out number))
        {
            Console.WriteLine("Введите число: ");
        }
        switch (number)
        {
            //отгадай ответ
            case 1:
                double A;
                double B;
```

```
                Console.WriteLine(
                     "F=((Sin^2(A)+Cos^3(B))/Sin^3(A)-
                     Cos^2(B))^(1/2)");
                Console.WriteLine("Введите переменную
A: ");
                while (!dou-
ble.TryParse(Console.ReadLine(), out A))
                {
                    Console.WriteLine("Введите число:
");
                }
                Console.WriteLine("Введите переменную
B: ");
                while (!dou-
ble.TryParse(Console.ReadLine(), out B))
                {
                    Console.WriteLine("Введите число:
");
                }
//вычисление математической функции
    try
    {
        double result = Math.Sqrt((c + d) / (e - f));
        if (double.IsNaN(result))
            throw new Exception();
        Console.WriteLine("Результат (округлён): " +
                        Math.Round(result));
```

```
        if (Math.Round(result) == you)
            Console.WriteLine("Молодец! Твой ответ
совпал с правильным!");
        else
            Console.WriteLine("Твой ответ неправиль-
ный! Попытайся заново!");
    }
    catch (DivideByZeroException)
    {
        Console.WriteLine("Деление на 0!");
    }
    catch (Exception)
    {
        Console.WriteLine("Число под корнем отрица-
тельное!");
    }
    Console.ReadKey();
    Console.Clear();
    break;
case 2:
        Console.WriteLine("Фамилия Имя Отчество\n
                    группа XXXX-XXXXXXX");
        Console.ReadKey();
        Console.Clear();
        break;
    case 3:
```

```
            Console.WriteLine("Задание");
            Console.ReadKey();
            Console.Clear();
            break;
        case 4:
            Console.WriteLine("Выйти из программы? д/н ");
            string str = Console.ReadLine();
            if (str == "д")
                Environment.Exit(0);
            else
                if (str == "н")
                    Console.Clear();
                else
                    Console.WriteLine("Значение введено
неверно, повторите попытку");
            Console.ReadKey();
            Console.Clear();
            break;
        default:
            Console.WriteLine("Ошибка!\n Выбранное значе-
                        ние введено неверно, повторите
                        ввод");
            Console.ReadKey();
            Console.Clear();
            break;
    }
}
```

3.3 Демонстрация и тестирование программы

В таблице 4 представлены тесты для проверки работы программы.

Таблица 4 – Набор тестов

№	Входные данные	Результат
1	a=1; b=90	Деление на 0!
2	a=0; b=0	Число под корнем отрицательное!
3	a= a; b=	Введи число:

Результаты работы программы представлены на рисунках 4 и 5.

```
 1 - Выполнение лабораторной работы
 2 - Об авторе
 3 - Задание
 4 - Выход
1
F=((Sin^2(A)+Cos^3(B))/Sin^3(A)-Cos^2(B))^(1/2)
Введите переменную  A:
0
Введите переменную B:
0
Чему равно значение F=((Sin^2(A)+Cos^3(B))/Sin^3(A)-Cos^2(B))^(1/2)?
0
Число под корнем отрицательное!
```

Рисунок 4 – Результат теста № 2

```
 1 - Выполнение лабораторной работы
 2 - Об авторе
 3 - Задание
 4 - Выход
1
F=((Sin^2(A)+Cos^3(B))/Sin^3(A)-Cos^2(B))^(1/2)
Введите переменную  A:
a
Введите число:
```

Рисунок 5 – Результат теста № 3

3.4 Результаты

В результате выполнения лабораторной работы изучены принципы работы условных операторов в Visual Studio. Изучен метод TryParse, контролирующий ввода данных, а также конструкция try-catch-finally для обработки исключений, связанных с вычислением математической функции. Разработано и протестировано консольное приложение.

4 Лабораторная работа 3. Циклы

4.1 Задание

Зациклить выполнение программы из лабораторной работы 2. При неправильном вводе ответа более трёх раз сообщать пользователю о проигрыше. Программа должна закрываться только при выборе пункта меню «Выход».

4.2 Разработка программы

В лабораторной работе необходимо написать программу для циклического выполнения подсчета функции при ошибке менее трёх раз. Список переменных, необходимых для реализации программы, приведен в таблице 5.

Таблица 5 – Переменные программы

Имя	Семантика	Тип	Ограничения
n	Переменная цикла	integer	[0;3]

Текст цикла функции в главной программе:

```
case 1:
    double A;
    double B;
    Console.WriteLine(
"F=((Sin^2(A)+Cos^3(B))/Sin^3(A)-Cos^2(B))^(1/2)");
    int i = 1;
    double result = 0;
    do
    {
        i--;
        Console.WriteLine("Введите переменную  A: ");
```

```
        while (!double.TryParse(Console.ReadLine(),
out A))
        {
            Console.WriteLine("Введите число: ");
        }
        Console.WriteLine("Введите переменную B: ");
        while (!double.TryParse(Console.ReadLine(),
out B))
        {
            Console.WriteLine("Введите число: ");
        }
        double t = Math.Pow(Math.Sin(A), 2)+
Math.Pow(Math.Cos(B),3);
        double r = Math.Pow(Math.Sin(A), 3)-
Math.Pow(Math.Cos(B),2);
        if (r == 0)
        {
            Console.WriteLine("Деление на 0!");
            i++;
        }
        else
        {
            result = Math.Sqrt(t / r);
            if (double.IsNaN(result))
            {
                Console.WriteLine("Ошибка! Число под
корнем является отрицательным!");
                i++;
```

```
            }
        }
    }
    while (i == 1);
    int count = 1;
    do
    {
        Console.WriteLine("Чему равно значение
 F=((Sin^2(A)+Cos^3(B))/Sin^3(A)-Cos^2(B))^(1/2)?");
        double youranswer;
        while (!double.TryParse(Console.ReadLine(),
out youranswer))
        {
            Console.WriteLine("Введите число: ");
        }
        if (result == youranswer)
            Console.WriteLine("Молодец! Твой ответ
совпал с правильным!");
        else
            Console.WriteLine("Твой ответ неправиль-
ный! Попытайся заново!");
        count++;
    }
    while ((count < 4) && (true));
    Console.WriteLine("Ты проиграл!");
    Console.ReadKey();
    Console.Clear();
    break;
```

4.3 Демонстрация и тестирование программы

В таблице 6 представлены тесты для проверки работы программы.

Таблица 6 – Набор тестов

№	Входные данные	Результат
1	a=1; b=1; youranswer= 1	Твой ответ неправильный!
2	a=1; b=1; youranswer= 3	Твой ответ неправильный!
3	a=1; b=1; youranswer= 2	Твой ответ совпал с правильный!

Результат работы программы представлен на рисунке 6.

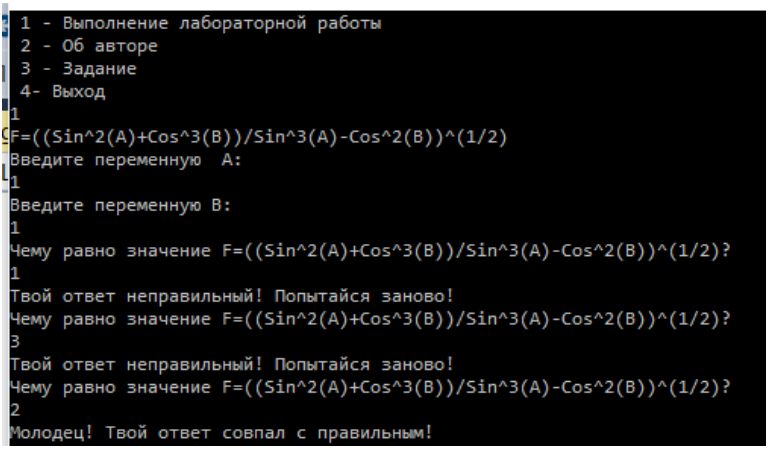

Рисунок 6 – Результат тестов № 1, 2, 3

4.4 Результаты

В результате выполнения лабораторной работы изучены принципы работы операторов цикла, их синтаксис и применение в Visual Studio. Разработано и протестировано консольное приложение.

5 Лабораторная работа 4. Работа с одномерными массивами

5.1 Задание

К программе из лабораторной работы 3 добавить пункт меню «Сортировка». Написать программу, сортирующую массив из n элементов двумя видами сортировки: сортировка Шелла, сортировка вставками. Сравнить время выполнения сортировок.

В программе должен быть реализован статический метод заполнения одномерного массива случайными значениями и статический метод вывода одномерного массива на экран.

5.2 Разработка программы

В лабораторной работе необходимо написать программу для сортировки одномерного массива. Для этого разработаем два метода сортировки массива InsSort и ShellSort. Список переменных, необходимых для реализации метода InsSort, приведен в таблице 7.

Таблица 7 – Переменные метода InsSort

Имя	Семантика	Тип	Ограничения
i	Переменная цикла	integer	[1;4]
j	Переменная цикла	integer	–
n	Длина массива	integer	–
a	"Ведро"	integer	–

Схема алгоритма метода InsSort представлена на рисунке 7.

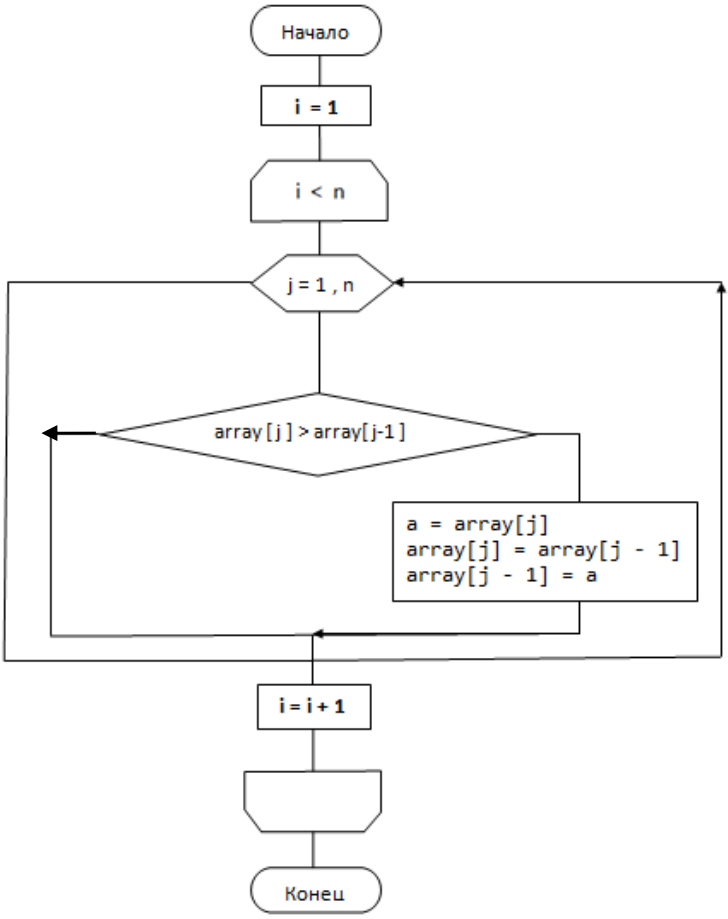

Рисунок 7 – Схема алгоритма метода InsSort

Текст метода InsSort для сортировки массива в порядке убывания:

```
static void InsSort (int[] array, int n)
```

```
{
    int i, a, number, j;
    i = 1;
    while (i < n)
    {
        for (j = 1; j < n; j++)
        {
            if (array[j] > array[j - 1])
            {
                a = array[j];
                array[j] = array[j - 1];
                array[j - 1] = a;
             }
        }
        i++;
    }
}
```

Текст метода Sort для обработки произвольного массива:

```
static void Sort(int n)
{
    int[] array = new int[n];
    RandomArray(array, n);
    OutputArray(array, n);
    int[] array2 = new int[n];
    array.CopyTo(array2, 0);
    Console.ReadKey();
```

```
        Console.WriteLine();
        Console.WriteLine("Сортировка Шелла");
        ShellSort(array, n);
        OutputArray(array, n);
        Console.WriteLine(" ");
        Console.WriteLine("Сортировка вставками");
        Ins(array2, n);
        OutputArray(array2, n);
        Console.ReadKey();
        Console.Clear();
}
```

5.3 Демонстрация и тестирование программы

В таблице 8 представлены тесты для проверки работы программы.

Таблица 8 – Набор тестов

№	Входные данные	Результат		
		Исх. массив	Сортировка Шелла	Сортировка вставками
1	n = 3	5 2 9	2 5 9	9 5 2
2	n = 5	6 -7 -3 9 -3	-7 -3 -3 6 9	9 6 -3 -3 -7

Результаты работы программы представлены на рисунках 8 и 9.

```
 1 - Выполнение лабораторной работы
 2 - Об авторе
 3 - Задание
 4 - Сортировка
 5 - Игра
 6 - Выход
4
Размер массива: 3
5  2  9
Сортировка Шелла
2  5  9
Сортировка вставками
9  5  2
```

Рисунок 8 – Результат тестов № 1

```
 1 - Выполнение лабораторной работы
 2 - Об авторе
 3 - Задание
 4 - Сортировка
 5 - Игра
 6 - Выход
4
Размер массива: 5
6  -7  -3  9  -3
Сортировка Шелла
-7  -3  -3  6  9
Сортировка вставками
9  6  -3  -3  -7
```

Рисунок 9 – Результат тестов № 2

5.4 Результаты

В результате выполнения лабораторной работы изучен алгоритм работы с одномерными массивами. Изучены методы классов Random, Array для ввода и вывода элементов массива и работы с ними. Разработано и протестировано консольное приложение.

6 Лабораторная работа 5. Работа с двумерными массивами

6.1 Задание

К программе из лабораторной работы 4 добавить пункт меню «Игра». В программе должен быть реализован статический метод заполнения массива и статический метод ввода массива с прорисовкой игрового поля. Создать двумерный массив размером 8x8, состоящий из 0 – поле игры в шахматы. Случайно образом поставить на поле ладью цифрой 1. Напишите программу, которая отмечает все поля, которые бьёт ладья цифрой 2.

6.2 Разработка программы

В лабораторной работе необходимо написать программу для создания двухмерного массива в виде шахматного поля. Для этого разработаем метод Game для заполнения массива и ввода его в виде игрового поля. Список переменных для реализации метода приведен в таблице номер 9.

Таблица 9 – Переменные метода Game

Имя	Семантика	Тип	Ограничения
n	Переменная цикла	integer	[1; 4]
ground[,]	Матрица поля	array	[n, n]
i , j	Переменные цикла	integer	
k, t	Произвольная строка/столбец	integer	

Текст метода Game для произвольного заполнения массива и его вывода:

```
static void Game()
{
    Console.WriteLine("Ход ладьи(1) с рандомного ме-
ста");
    int n = 8;
    int[,] ground = new int[n, n];
    Random rnd = new Random();
    for (int i = 0; i < n; i++)
    {
        for (int j = 0; j < n; j++)
            ground[i, j] = 0;
    }
    int k = rnd.Next(0, n);
    int t = rnd.Next(0, n);
    for (int j = 0; j < n; j++)
        ground[k, j] = 2;
    for (int i = 0; i < n; i++)
        ground[i, t] = 2;
    ground[k, t] = 1;
    for (int i = 0; i < n; i++)
    {
        Console.WriteLine();
        for (int j = 0; j < n; j++)
            Console.Write("{0} ", ground[i, j]);
```

```
        }
        Console.ReadKey();
        Console.Clear();
}
```

6.3 Демонстрация и тестирование программы

Результат работы программы представлен на рисунке 10.

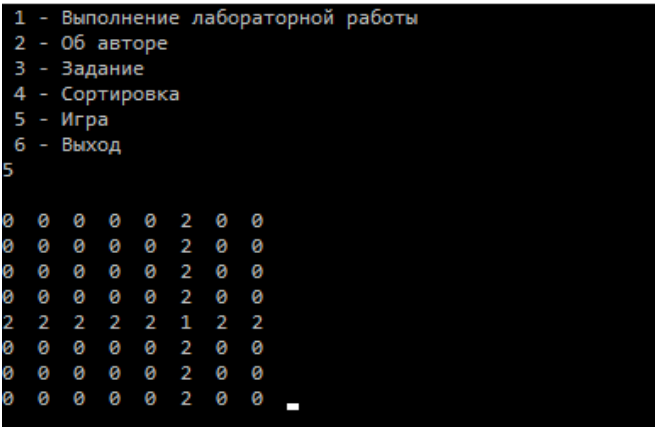

Рисунок 10 – Результат теста

6.4 Результаты

В результате выполнения лабораторной работы изучен алгоритм работы с многомерными массивами. Изучены методы классов Random, Array для ввода и вывода элементов массива и работы с ними. Разработано и протестировано консольное приложение.

7 Лабораторная работа 6. Работа со строками

7.1 Задание

Добавить в программу пункт меню для работы со строками. Предусмотреть работу с тестовыми строками и работу со строками, которые вводит пользователь. Посчитать количество цифр в исходной строке.

Строка 1:
Вот 1 (иль единица),
Очень тонкая, как спица,
А вот это цифра 2.
Полюбуйся, какова:
Выгибает двойка шею,
Волочится хвост за нею.
А за двойкой – посмотри –
Выступает цифра 3.
Тройка – третий из значков –
Стоит из двух крючков.

7.2 Разработка программы

В лабораторной работе необходимо написать программу для подсчёта количества цифр в строке. Для этого разработаем метод

подсчёта количества цифр в строке WorkString. Список переменных, необходимых для реализации метода приведен в таблице 10.

Таблица 10 – Переменные метода WorkString

Имя	Семантика	Тип	Ограничения
s	Строка с искомыми значениями	string	{1, ..., 9, 0}
s1	Рабочая строка	string	
n	Переменная выбора	integer	{1, 2}
i	Переменная цикла	integer	
h	Счётчик	integer	

Текст метода WorkString для подсчёта количества цифр в строке:

```
static void WorkString()
{
    char [] Cnt = { '0', '1', '2', '3', '4', '5', '6',
'7', '8', '9' };
    int n=0;
    Console.WriteLine(" 1. Обработка исходной строки\n
2. Ввести свою строку для обработки");
    while ((n!= 1) && (n!=2))
    {
        while (!int.TryParse(Console.ReadLine(), out
n))
```

```
        {
            Console.WriteLine("Введите число: ");
        }
        string s1 = "";
        if (n == 1)
        {
            s1 = "Исходная строка";
            Console.WriteLine(s1);
        }
        else if (n == 2)
        {
            s1 = Console.ReadLine();
        }
        else
        {
            Console.WriteLine("Введите цифру, нужного
задания(1 или 2)");
         }
         int h = 0;
         for (int i = 0; i < s1.Length; i++)
             for (int j = 0; j < 9; j++)
                 if (s1[i] == Cnt[j])
                     h++;
         Console.WriteLine("Количество цифр в строке:
" + h);
         Console.ReadKey();
```

```
            Console.Clear();
        }

}
```

7.3 Демонстрация и тестирование программы

В качестве теста для проверки работы программы возьмём произвольную строку «2 мамы мыли 100 рам». В результате получим число 4 (количество цифр в строке).

Результат работы программы представлена на рисунке 11.

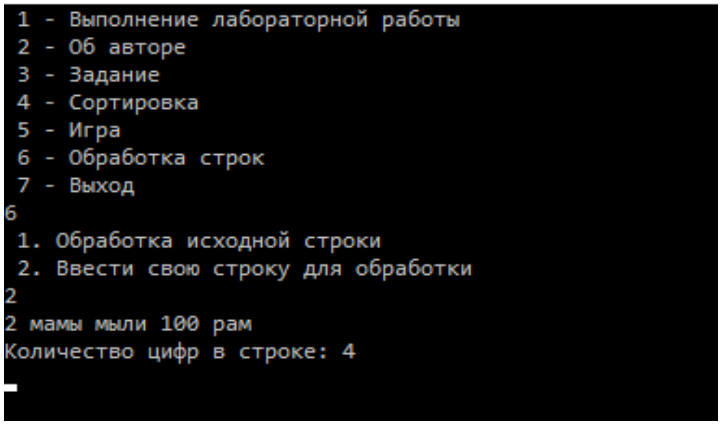

Рисунок 11 – Результат теста

7.4 Результаты

В результате выполнения лабораторной работы изучен принцип обработки строк. Изучены методы типа Char и класса String для ввода и вывода данных и работы с текстовыми строками. Разработано и протестировано консольное приложение.

8 Лабораторная работа 7. Классы

8.1 Задание

Реализовать лабораторную работу в виде классов с XML-документированием.

8.2 Разработка программы

В лабораторной работе необходимо написать программу в виде классов с XML-документированием. Для этого разобьём лабораторную работу по классам, в каждом из которых будут представлены отдельные методы.

Текст метода Rook для подсчета суммы цифр числа:

```
    /// <summary>
    /// Класс ладьи
    /// </summary>
    /// <permission cref="_n">Размер
массива</permission>
    public class Rook
    {
        private readonly int _n = 8;
        public void Play()
        {
            Console.WriteLine("Ход ладьи(1) с рандом-
ного места");
            //  Размер поля (шахматной доски)
            int n=_n;
```

```
            int[,] ground = new int[n, n];
            Random rnd = new Random();
            //Визуализация клеток поля
            for (int i = 0; i < n; i++)
            {
                for (int j = 0; j < n; j++)
                    ground[i, j] = 0;
            }
            // Визуализация хода ладьи по столбцу и
строке
            int k = rnd.Next(0, n);
            int t = rnd.Next(0, n);
            for (int j = 0; j < n; j++)
                ground[k, j] = 2;
            for (int i = 0; i < n; i++)
                ground[i, t] = 2;
            //Постановка фигуры-ладьи
            ground[k, t] = 1;
            for (int i = 0; i < n; i++)
            {
                Console.WriteLine();
                for (int j = 0; j < n; j++)
                    Console.Write("{0}  ", ground[i,
j]);
            }
        }
    }
```

Данный класс содержит приватное поле _n, задающее размеры массива–шахматного поле; класс Play, реализующий построение игрового поля и произвольную постановку ладьи с направлением её возможных ходов.

8.3 Результаты

В результате выполнения лабораторной работы изучены основные принципы объектно-ориентированного программирования в Visual Studio. Изучены различные члены класса и модификаторы доступа. Программа разбита на классы и объекты с XML-документированием. Разработано и протестировано консольное приложение.

9 Лабораторная работа 8. Создание приложения WinForms

9.1 Задание

Создать проект приложения Windows Forms, разработать формы и реализовать логику для проекта из лабораторной работы 7.

9.2 Разработка программы

В лабораторной работе необходимо создать проект на основе лабораторной работы 7. Для этого разработаем форму Lab8. С помощью элемента TabContol создадим коллекцию вкладок для различных элементов управления программы. Создадим в форме кнопки для работы и выполнения частей программы.

Текст события Click кнопки ButtonRook для визуализации хода произвольной ладьи:

```
public void ButtonRook_Click(object sender, EventArgs e)
{
    CreatBoard();
    int n = 8;
    Random rnd = new Random();
    int k = rnd.Next(0, n);
    int t = rnd.Next(0, n);
    for (int j = 0; j < 8; j++)
    {
        dataGridViewRook.Rows[k].Cells[j].Style.BackColor =
Color.LightGreen;
    }
    for (int i = 0; i < 8; i++)
```

```
    {
        dataGridViewRook.Rows[i].Cells[t].Style.BackColor =
Color.LightGreen;
    }
    dataGridViewRook.Rows[k].Cells[t].Style.BackColor =
Color.Red;
}
```

Данный метод окрашивает в нужный цвет ячейки поля, визуализируя тем самым ход произвольной ладьи.

9.3 Демонстрация и тестирование программы

Результаты работы программы представлены на рисунках 12-15.

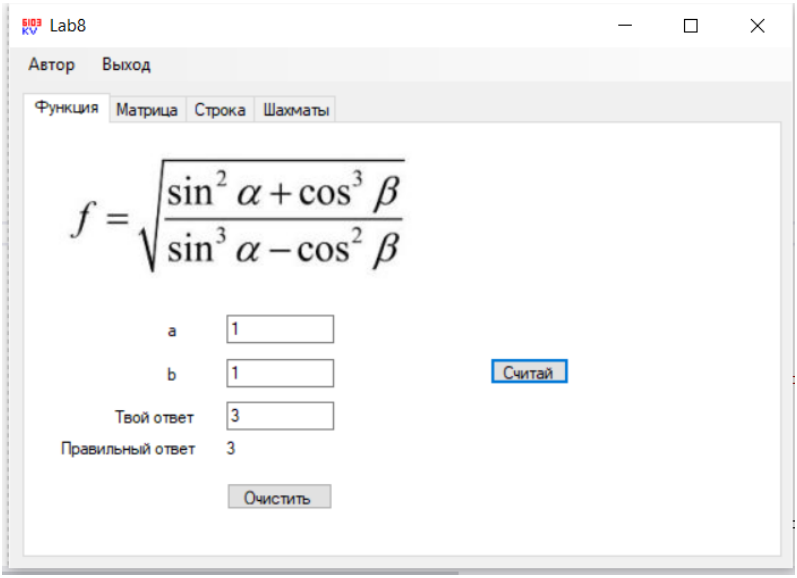

Рисунок 12 – Результат теста № 1

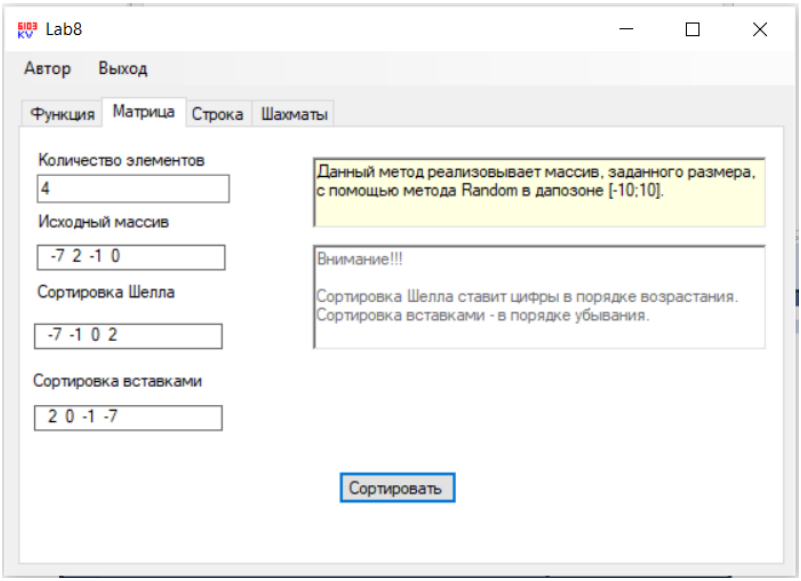

Рисунок 13 – Результат теста № 2

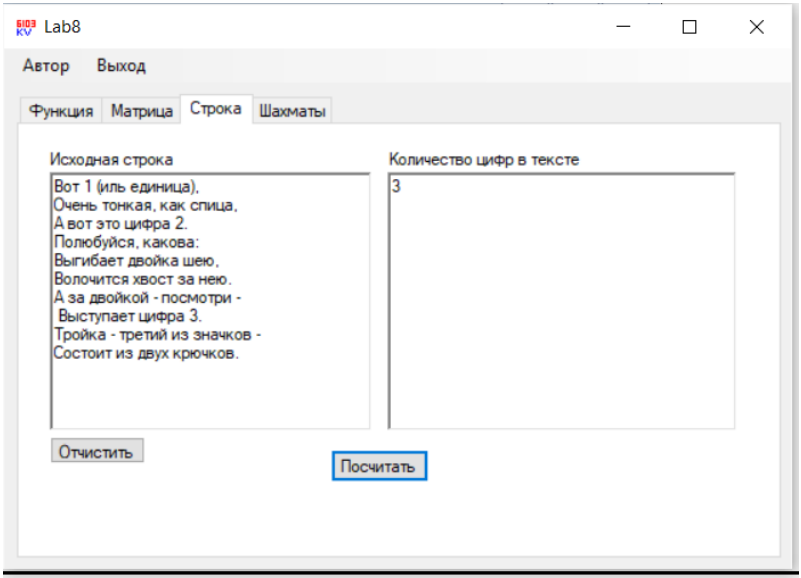

Рисунок 14 – Результат теста № 3

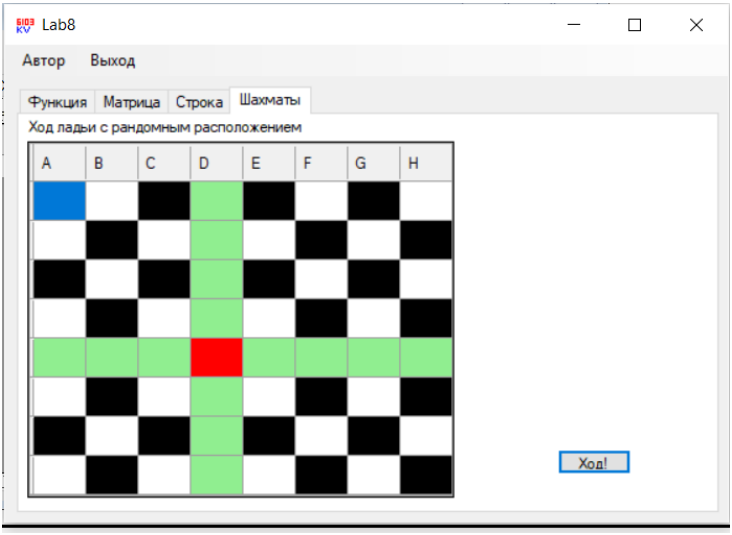

Рисунок 15 – Результат теста № 4

9.4 Результаты

В результате выполнения лабораторной работы изучены основы работы с Windows Forms: изучены элементы, их свойства и события, а также наборы инструментов для создания форм. Разработано и протестировано приложение Windows Forms.

СПИСОК ИСПОЛЬЗОВАННЫХ ИСТОЧНИКОВ

1 ГОСТ 2.105-95 Единая система конструкторской документации. Общие требования к текстовым документам [Текст]. – Введ. 1996-06-30. – М: Издательство стандартов, 1995. – 31 с.

2 Павловская, Т.А. C#. Программирование на языке высокого уровня. Учебник для вузов [Текст] / Т.А. Павловская. – СПб.: Питер, 2007. – 432 с.

3 СТО 02068410-004-2018 Общие требования к учебным текстовым документам [Текст]. – Самара: Самарский университет, 2018. – 36 с.

ПРИЛОЖЕНИЕ Б

Контрольные вопросы

1) Какие типы данных относятся к целочисленным?
   а) long
   б) float
   в) short
   г) decimal
   д) string
   е) int

2) Какие типы данных являются знаковыми?
   а) sbyte
   б) float
   в) long
   г) byte
   д) double
   е) ushort
   ж) short
   з) uint
   и) ulong
   к) decimal
   л) int
   м) char

3) Какие типы данных являются беззнаковыми?
   а) long
   б) ushort
   в) byte
   г) short
   д) decimal

е) double
ж) uint

4) Какие типы данных являются типами с плавающей точкой?
а) int
б) float
в) byte
г) long
д) bool
е) short
ж) double

5) Какой тип данных предназначен для использования в денежных вычислениях?
а) float
б) decimal
в) double
г) byte
д) int

6) Какие типы данных могут использоваться в качестве операндов логических операторов?
а) bool
б) int
в) float
г) string

7) В каких случаях значение переменной x увеличится на 20?
а) x := x + 20;
б) x +=20;
в) x = x + 20;
г) x == x + 20;

8) Какие значения может принимать тип bool?
   а) true
   б) 1
   в) false
   г) 0

9) Какой оператор возвращает остаток от деления?
   а) /
   б) div
   в) %
   г) &&

10) Какой оператор является быстрым логическим «ИЛИ»?
   а) |
   б) &&
   в) //
   г) /
   д) &
   е) ||

11) Какие операторы относятся к группе логических «И»?
   а) |
   б) &&
   в) ||
   г) ^
   д) !
   е) &

12) Какой результат выполнения фрагмента кода?

```
long l = 1234;
double d = l;
Console.WriteLine(d * d);
```

Console.ReadKey();

а) Код не выполнится, нужно использовать явное преобразование типов.
б) 1522756
в) Код не выполнится даже с явным преобразованием типов.

13) Выполнится ли следующий фрагмент кода?
int f = 1;
bool flag = f;

а) Да, переменная flag примет значение true.
б) Нет, т.к. нужно выполнить явное преобразование типов.
в) Нет, т.к. типы int и bool не совместимы.
г) Да, переменная flag примет значение 1.

14) Какой оператор является быстрым логическим «И»?
а) &
б) |
в) ||
г) /
д) &&
е) //
ж) !

15) Какие операторы относятся к группе логических «ИЛИ»?
а) &
б) ||
в) \\
г) |
д) ^
е) //
ж) &&

16) Какой оператор является логическим отрицанием?

а) &
б) |
в) &&
г) ||
д) ^
е) !

17) Какой результат выполнения фрагмента кода?

```
double d = 1234;
long l = d;
Console.WriteLine(l * l);
Console.ReadKey();
```

а) Код не выполнится, нужно использовать явное преобразование типов.
б) 1522756
в) Код не выполнится даже с явным преобразованием типов.

18) Какие выражения записаны неверно?

а) x +=20;
б) x := x + 20;
в) x = x + 20;
г) x == x + 20;

19) Является ли C# регистрозависимым языком?

д) да
е) нет

20) Для чего нужен метод Parse?

а) Возвращает строковый массив, содержащий подстроки, разделенные заданными элементами.
б) Выполняет преобразование строки в тот тип данных, который был целью вызова метода.

в) Вырезает определенную часть строки.
г) Изменяет регистр.

21) Какой оператор является исключающим «ИЛИ»?
а) |
б) ^
в) &&
г) ||
д) \\
е) &

22) В каких случаях значение переменной x уменьшится на 15?
а) x -= 15;
б) x = x - 15;
в) x := x - 15;
г) x == x - 15;

23) Как можно назвать переменную?
а) count
б) 5x
в) _max
г) min5
д) for

24) Какой результат выполнения фрагмента кода?
```
double x = 1.234;
decimal y = x;
Console.WriteLine(2 * y);
Console.ReadKey();
```
а) Код не выполнится, нужно использовать явное преобразование типов.
б) 2,468
в) Код не выполнится даже с явным преобразованием типов.

25) Какие символы используются для обозначения комментариев?

а) \\ Это комментарий
б) // Это комментарий
в) /* Это комментарий */
г) “ Это комментарий ”
д) ‘ Это комментарий ‘

26) Где допущена ошибка?

```
int a = 15;
int b = 20;
return A + b;
```

а) Нигде.
б) В операторе return нельзя использовать математические выражения.
в) Выражения в операторе return нужно записывать в скобках.
г) Переменная A не существует.

27) Чем отличаются методы Write и WriteLine?

а) Ничем
б) Метод WriteLine выводит текст на консоль. Метод Write выводит текст на консоль, после чего осуществляет переход на новую строку.
в) Метод Write выводит текст на консоль. Метод WriteLine выводит текст на консоль, после чего осуществляет переход на новую строку.
г) Метод Write выводит текст на консоль посимвольно. Метод WriteLine выводит весь текст на консоль.

28) Какие операторы являются операторами цикла?

а) while
б) if

в) do-while
г) ?:
д) foreach
е) switch
ж) for

29) Какой результат выполнения фрагмента кода?

```
int count = 3;
double price = 1500;
if (count == 3)
{
   double sum = 2 * price;
}
Console.WriteLine("Сумма покупки {0} рублей", sum);
```

а) Код не выполнится, т.к. в теле условного оператора нельзя объявлять переменные.
б) Сумма покупки 3000 рублей.
в) Код не выполнится, т.к. метод WriteLine принимает только один параметр.
г) Код не выполнится, т.к. переменная sum вне зоны области видимости.

30) Какой тип данных может принимать условное выражение, управляющее оператором if?

а) string
б) int
в) bool
г) double

31) Является ли часть else оператора if обязательной?

а) да
б) нет

32) Какой тип данных запрещен в выражении, управляющем оператором switch?

а) int
б) double
в) string
г) char

33) Какой результат выполнения фрагмента кода?

```
public enum Dessert
{
   Eclair = 0,
   Donut = 1,
   Cake = 2
}

class Program
{
   static void Main(string[] args)
   {
      Dessert d = Dessert.Cake;
      switch (d)
      {
         case Dessert.Cake:
         case Dessert.Cake:
            Console.WriteLine("The dessert is cake");
            break;
         case Dessert.Eclair:
            Console.WriteLine("The dessert is eclair");
            break;
         default:
            Console.WriteLine("The dessert is unknown.");
            break;
      }
```

```
    }
}
```

а) Код не выполнится, т.к. метки case должны быть уникальны.
б) Код не выполнится, т.к. разделы case нельзя оставлять пустыми.
в) Будет выведена фраза "The dessert is cake".

34) Какой результат выполнения фрагмента кода?

```
int x = 0;
Console.WriteLine(x == 0 ? (1.0 / x).ToString("f4") : "Деление на 0.");
```

а) Деление на 0.
б) 0,5000.
в) Программа не скомпилируется.
г) Бесконечность.
д) Nan

35) Какие операторы являются условными?

а) while
б) foreach
в) switch
г) ?:
д) do-while
е) if
ж) for

36) Какой результат выполнения фрагмента кода?

```
int count = 1;
double price = 1500;
while (count < 3)
{
```

```
    double sum = count * price;
    count++;
}
Console.WriteLine("Сумма покупки {0} рублей", sum);
Console.ReadKey();
```

а) Код не выполнится, т.к. переменная sum вне зоны области видимости.

б) Код не выполнится, т.к. в теле цикла нельзя объявлять переменные.

в) Сумма покупки 3000 рублей

г) Сумма покупки 4500 рублей

37) Можно ли использовать часть if без части else?

а) да

б) нет

38) Какое значение должно вернуть условное выражение оператора if, чтобы выполнился блок операторов, принадлежащий части else?

а) 0

б) 1

в) true

г) false

39) Сколько раз выполнится цикл?

```
int x = 5;
do
{
    Console.WriteLine(x);
    x++;
} while (x < 5);
```

а) 0

б) 1

в) 2

40) Может ли отсутствовать ветвь default у оператора switch?

а) да

б) нет

41) Какой результат выполнения фрагмента кода?

```
public enum Dessert
{
   Eclair = 0,
   Donut = 1,
   Cake = 2
}
class Program
{
   static void Main(string[] args)
   {
      Dessert d = (Dessert)(new Random()).Next(0, 3);
      switch (d)
      {
         case 0:
            Console.WriteLine("The dessert is cake");
            break;
         case 1:
            Console.WriteLine("The dessert is donut");
            break;
         case 2:
            Console.WriteLine("The dessert is eclair");
            break;
         default:
            Console.WriteLine("The dessert is unknown.");
            break;
      }
   }
}
```

а) Выполнится код в одной из меток case в зависимости от выпавшего значения d.
б) Код не выполнится, т.к. тип выражения соответствия не совпадает с типом метки case.
в) Выполнится раздел default.

42) Какой результат выполнения фрагмента кода?

```
int x = 0;
Console.WriteLine(x == 0 ? "Деление на 0." : (1.0 / x).ToString("f4"));
```

а) Деление на 0.
б) 0,5000.
в) Программа не скомпилируется.
г) Бесконечность.
д) Nan

43) Какие способы объявления массива являются корректными?

а) int[] myArray = new int[] { 1, 3, 5, 7, 9 };
а) int[] myArray = new int[5];
б) int myArray[] = new int[5];
в) int[] myArray = { 1, 2, 3, 4, 5, 6 };
г) int[5] myArray = new int[5];

44) К какому модификатору доступа относится следующее утверждение: доступ к защищенному элементу может быть получен из соответствующего класса, а также экземплярами производных классов?

а) public
б) internal
в) private
г) protected

45) Полиморфизм – это

а) способ выделить набор значимых характеристик объекта, исключая из рассмотрения незначимые.

б) механизм программирования, который связывает действия и данные, которыми он манипулирует, и при этом предохраняет их от вмешательства извне и неправильного использования.

в) свойство системы использовать объекты с одинаковым интерфейсом без информации о типе и внутренней структуре объекта.

г) процесс, благодаря которому один объект может приобретать свойства другого.

46) Какие утверждения верны для абстрактных классов?

а) Абстрактные классы могут определять абстрактные методы.

б) Абстрактные классы предотвращают наследование.

в) Создавать экземпляры абстрактного класса нельзя.

г) Классы, производные от абстрактного класса, должны реализовывать все абстрактные методы.

47) Какие способы объявления двумерного массива являются корректными?

а) int[,] myArray = new int[5, 3]

б) int[5, 3] myArray = new int[,];

в) int[] myArray = {{1, 2}, {3, 4}};

г) int[,] myArray = new int[,] {{1, 2}, {3, 4}};

48) Какой модификатор доступа объявлен по умолчанию для членов класса?

а) public

б) internal

в) private

г) protected

49) Инкапсуляция – это

а) способ выделить набор значимых характеристик объекта, исключая из рассмотрения незначимые.

б) механизм программирования, который связывает действия и данные, которыми он манипулирует, и при этом предохраняет их от вмешательства извне и неправильного использования.

в) свойство системы использовать объекты с одинаковым интерфейсом без информации о типе и внутренней структуре объекта.

г) процесс, благодаря которому один объект может приобретать свойства другого.

50) Какие уровни доступа могут иметь члены класса?

а) internal

б) protected

в) protected internal

г) internal private

51) Какие утверждения верны для запечатанных классов?

а) Запечатанный класс не может быть абстрактным классом.

б) Создавать экземпляры запечатанного класса нельзя.

в) Члены запечатанного класса должны быть реализованы в производном класс.

г) Запечатанный класс не может использоваться в качестве базового класса.

52) Назовите основные принципы ООП.

53) Напишите свой метод поиска индекса последнего вхождения заданного числа в одномерный массив.

54) Напишите свой метод поиска индекса первого вхождения заданного числа в одномерный массив.

55) Какой метод класса Array возвращает индекс последнего вхождения заданного числа в одномерный массив?

56) Какой метод класса Array возвращает индекс первого вхождения заданного числа в одномерный массив?

57) Написать программу для вычисления суммы чисел от 100 до 200 кратных 17.

58) Написать программу для вычисления произведения двузначных нечетных чисел кратных 13.

Учебное издание

Столбова Анастасия Александровна,
Головнин Олег Константинович

ТЕОРЕТИЧЕСКИЕ ОСНОВЫ И ПРАКТИЧЕСКИЕ АСПЕКТЫ ИНФОРМАТИКИ И ПРОГРАММИРОВАНИЯ ДЛЯ РЕШЕНИЯ ЗАДАЧ УПРАВЛЕНИЯ И ОБРАБОТКИ ИНФОРМАЦИИ НА ЯЗЫКЕ C#

Учебное пособие

В авторской редакции

Компьютерная верстка Л.Р. Дмитриенко

Подписано в печать 17.09.2019. Формат 60x84 1/16.
Бумага офсетная. Печ. л. 10,25.
Тираж 50 экз. Заказ .

ФЕДЕРАЛЬНОЕ ГОСУДАРСТВЕННОЕ АВТОНОМНОЕ
ОБРАЗОВАТЕЛЬНОЕ УЧРЕЖДЕНИЕ ВЫСШЕГО ОБРАЗОВАНИЯ
«САМАРСКИЙ НАЦИОНАЛЬНЫЙ ИССЛЕДОВАТЕЛЬСКИЙ
УНИВЕРСИТЕТ ИМЕНИ АКАДЕМИКА С.П. КОРОЛЕВА»
(САМАРСКИЙ УНИВЕРСИТЕТ)
443086 Самара, Московское шоссе, 34.

Изд-во Самарского университета.
443086 Самара, Московское шоссе, 34.