

**МИНИСТЕРСТВО ОБРАЗОВАНИЯ И НАУКИ  
РОССИЙСКОЙ ФЕДЕРАЦИИ**  
Федеральное государственное бюджетное образовательное  
учреждение высшего профессионального образования  
**МОСКОВСКИЙ ГОСУДАРСТВЕННЫЙ ТЕХНИЧЕСКИЙ  
УНИВЕРСИТЕТ «МАМИ»**

В.И. Антомони  
В.Н. Архипов  
А.Н. Любин  
В.Н. Тихомиров

**«Основы программирования на С#»**

**Сборник лабораторных работ**  
по дисциплине «Информатика» и «Информационные технологии»  
для студентов  
всех направлений и специальностей

**Москва  
2011**

УДК 681.3.06

*Разработано в соответствии с Государственным образовательным стандартом 2008 г. Для всех направлений и специальностей на основе примерной программы дисциплины «Информатика»*

Рецензенты: генеральный директор «Института информационных технологий», д.т.н., профессор В. Г. Зубков;  
доцент кафедры «Автоматика и процессы управления»  
к.т.н. Ю. А. Савостьянок;

### **Основы программирования на С#**

Сборник лабораторных работ по дисциплине «Информатика» и «Информационные технологии» для студентов всех направлений и специальностей: М., МАМИ, 2011

Кафедра

«Информационные системы и дистанционные технологии»

Пособие ориентировано на изучение основ языка программирования С#, освоение программирования на этом языке и получение навыков решения задач на ПК.

Лабораторные работы № 1, 2 написаны В. Н. Архиповым, правила выполнения работ и лабораторные работы № 3, 4 написаны А. Н. Любиным, лабораторные работы № 5, 6 написаны В. И. Антомони, введение и лабораторная работа №7, написаны В. Н. Тихомировым.

© В.И. Антомони, В.Н. Архипов,  
А.Н. Любин, В.Н. Тихомиров  
© «МАМИ», 2011

# **ВВЕДЕНИЕ.**

## **Платформа .NET**

Программист пишет программу, компьютер ее выполняет. Программа создается на языке, понятном человеку, а компьютер умеет исполнять только программы, написанные на его языке – в машинных кодах.

Совокупность средств, с помощью которых программы пишут, корректируют, преобразуют в машинные коды, отлаживают и запускают, называют средой разработки.

Среда разработки обычно содержит:

- текстовый редактор, предназначенный для ввода и корректировки текста программы;
- компилятор, с помощью которого программа переводится с языка, на котором она написана, в машинные коды;
- средства отладки и запуска программ;
- общие библиотеки, содержащие многократно используемые элементы программ;
- справочную систему и другие элементы.

Платформа .NET (произносится «дот нэт») включает не только среду разработки для нескольких языков программирования, называемую Visual Studio.NET, но и множество других средств, например, механизмы поддержки баз данных, электронной почты и коммерции необходимых для интенсификации труда программиста.

Важнейшими задачами при создании программ в настоящее время становятся:

- переносимость — возможность выполнения на различных типах компьютеров;
- безопасность — невозможность несанкционированных действий;
- надежность — способность выполнять необходимые действия в определённых условиях.
- использование готовых компонентов — для ускорения разработки;
- межъязыковое взаимодействие — возможность применять одновременно несколько языков программирования.

Платформа .NET позволяет успешно решать все эти задачи. Для обеспечения переносимости компиляторы, входящие в состав платформы, переводят программу не в машинные коды, а в промежуточный язык MSIL (Microsoft Intermediate Language), или просто IL), который не содержит команд, зависящих от языка, операционной системы и типа компьютера. Программа на этом языке выполняется не самостоятельно, а под управлением системы, которая называется общезыковой средой выполнения (Common Language Runtime, CLR).

Среда CLR может быть реализована для любой операционной системы. При выполнении программы CLR вызывает так называемый JIT-компилятор, переводящий код с языка IL в машинные команды конкретного процессора, которые немедленно выполняются. JIT означает «just in time», что можно перевести как «во время», то есть компилируются только те части программы, которые требуется выполнить в данный момент. Каждая часть про-

граммы компилируется один раз и сохраняется в памяти для дальнейшего использования.

Схема выполнения программы при использовании платформы .NET приведена на рис. 1.

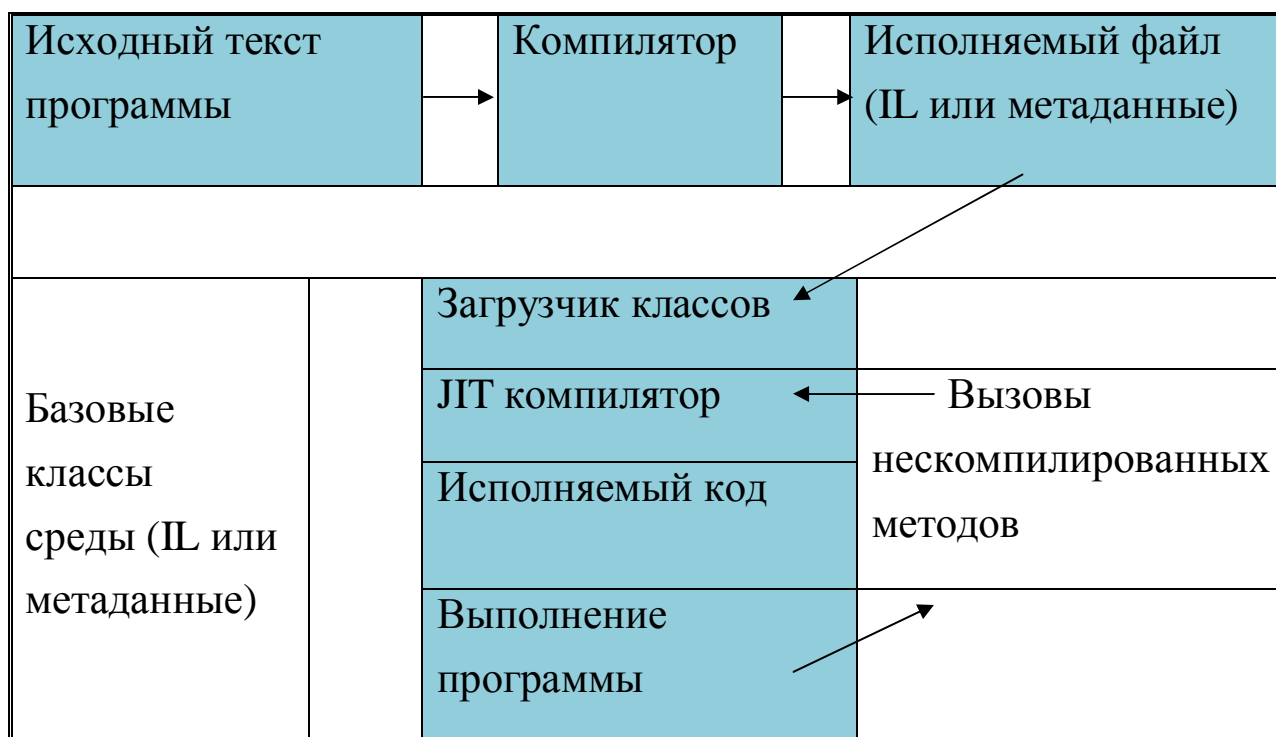


Рис. 0.1. Схема выполнения программы в .NET

Компилятор в качестве результата своего выполнения создает так называемую сборку – файл с расширением exe или dll, который содержит код на языке IL и метаданные. Метаданные представляют собой сведения об объектах, используемых в программе, а также сведения о самой сборке. Они позволяют организовать межъязыковое взаимодействие, обеспечивают безопасность и облегчают развертывание приложений, то есть установку программ на компьютеры пользователей.

Во время работы программы среда CLR следит за тем, чтобы выполнялись только разрешенные операции, осуществляет распре-

деление и очистку памяти и обрабатывает возникающие ошибки. Это многократно повышает безопасность и надежность программ.

Платформа .NET содержит огромную библиотеку классов, которые можно использовать при программировании на любом языке .NET. Её изучение – трудоёмкая, но необходимая задача.

Платформа .NET рассчитана на объектно-ориентированную технологию создания программ, поэтому прежде чем начинать изучение языка C#, необходимо познакомиться с основными понятиями объектно-ориентированного программирования (ООП).

Принципы ООП проще всего понять на примере программ моделирования. В реальном мире каждый предмет или процесс обладает набором статических и динамических характеристик, иными словами, свойствами и поведением. Поведение объекта зависит от его состояния и внешних воздействий. Например, объект «автомобиль» никуда не поедет, если в баке нет бензина, а если повернуть руль, изменится положение колес. Объект представляется как совокупность данных, характеризующих его состояние, и функций их обработки, моделирующих его поведение. Вызов функции на выполнение часто называют посылкой сообщения объекту.

При создании объектно-ориентированной программы предметная область представляется в виде совокупности объектов. Выполнение программы состоит в том, что объекты обмениваются сообщениями. Это позволяет использовать при программировании понятия, более адекватно отражающие предметную область.

Для моделирования реального объекта с помощью программного обеспечения выделяют его существенные особенности. Их набор зависит от цели моделирования. Выделение существенных с

той или иной точки зрения свойств называется абстрагированием. Таким образом, программный объект – это абстракция.

Важным свойством объекта является его обособленность. Детали реализации объекта, то есть внутренние структуры данных и алгоритмы их обработки, скрыты от пользователя объекта и недоступны для непреднамеренных изменений. Объект используется через его интерфейс – совокупность правил доступа.

Скрытие деталей реализации называется инкапсуляцией (от слова «капсула»). Таким образом, объект является «черным ящиком», замкнутым по отношению к внешнему миру. Это позволяет представить программу в укрупненном виде – на уровне объектов и их взаимосвязей, а следовательно, управлять большим объемом информации и успешно отлаживать сложные программы.

Сказанное можно сформулировать более кратко и строго: объект – это инкапсулированная абстракция с четко определенным интерфейсом.

Инкапсуляция позволяет изменить реализацию объекта без модификации основной части программы, если его интерфейс остался прежним. Простота модификации является очень важным критерием качества программы, ведь любой программный продукт в течение своего жизненного цикла претерпевает множество изменений и дополнений. Кроме того, инкапсуляция позволяет использовать объект в другом окружении и быть уверенным, что он не испортит не принадлежащие ему области памяти, а также создавать библиотеки объектов для применения во многих программах.

В мире пишется огромное количество новых программ, и важнейшее значение приобретает возможность многократного использования кода. Преимущество объектно-ориентированного програм-

мирования состоит в том, что для объекта можно определить наследников, корректирующих или дополняющих его поведение. При этом нет необходимости не только повторять исходный код родительского объекта, но даже иметь к нему доступ.

Наследование является мощнейшим инструментом ООП и применяется для следующих взаимосвязанных целей:

- исключения из программы повторяющихся фрагментов кода;
- упрощения модификации программы;
- упрощения создания новых программ на основе существующих.

Благодаря наследованию появляется возможность использовать объекты, исходный код которых недоступен, но в которые требуется внести изменения. Наследование позволяет создавать иерархии объектов. Иерархия представляется в виде дерева, в котором более общие объекты располагаются ближе к корню, а более специализированные – на ветвях и листьях. Наследование облегчает использование библиотек объектов, поскольку программист может взять за основу объекты, разработанные кем-то другим, и создать наследников с требуемыми свойствами.

Объект, на основании которого строится новый объект, называется родительским объектом, объектом-предком, базовым классом, или суперклассом, а унаследованный от него объект – потомком, подклассом, или производным классом.

ООП позволяет писать гибкие, расширяемые и читабельные программы. Во многом это обеспечивается благодаря полиморфизму, под которым понимается возможность во время выполнения программы с помощью одного и того же имени выполнять разные действия или обращаться к объектам разного типа.



Итак, объект это «инкапсуляция множества операций (методов), доступных для внешних вызовов, и состояния, запоминающего результаты выполнения указанных операций».

### **Достоинства ООП:**

- использование при программировании понятий, близких к предметной области;
- возможность успешно управлять большими объемами исходного кода благодаря инкапсуляции, то есть скрытию деталей реализации объектов и упрощению структуры программы;
- возможность многократного использования кода за счет наследования;
- сравнительно простая возможность модификации программ;
- возможность создания и использования библиотек объектов.

Однако создание объектно-ориентированной программы представляет собой весьма непростую задачу, поскольку требует разработки иерархии объектов, а плохо спроектированная иерархия может свести к нулю все преимущества объектно-ориентированного подхода. Кроме того, идеи ООП не просты для понимания и в особенности для практического применения. Чтобы эффективно использовать готовые объекты из библиотек, необходимо освоить большой объем достаточно сложной информации. Неграмотное же применение ООП способно привести к созданию излишне сложных программ, которые невозможно отлаживать и усовершенствовать.

Для представления объектов в языках C#, Java, C++, Delphi и др. используется понятие класс, аналогичное обыденному смыслу этого слова в контексте «класс членистоногих», «класс млекопитающих», «класс задач» и т. п. Класс является обобщенным поняти-

ем, определяющим характеристики и поведение некоторого множества конкретных объектов этого класса, называемых экземплярами класса. В последнее время в класс часто добавляется третья составляющая – события, на которые может реагировать объект класса.

Все классы библиотеки .NET, а также все классы, которые создает программист в среде .NET, имеют одного общего предка – класс `object` и организованы в единую иерархическую структуру. Внутри нее классы логически сгруппированы в так называемые пространства имен, которые служат для упорядочивания имен. Пространства имен могут быть вложенными, их идея аналогична знакомой вам иерархической структуре каталогов на компьютере.

Обычно в одно пространство имен объединяют взаимосвязанные классы. Например, пространство `System.Net` содержит классы, относящиеся к передаче данных по сети, `System.Windows.Forms` – элементы графического интерфейса пользователя, такие как формы, кнопки и т. д. Имя каждого пространства имен представляет собой неделимую сущность, однозначно его определяющую.

### **Среда Visual Studio.NET**

Среда разработки Visual Studio.NET предоставляет мощные и удобные средства написания, корректировки, компиляции, отладки и запуска приложений, использующих .NET-совместимые языки. Корпорация Microsoft включила в платформу средства разработки для четырех языков: C#, VB.NET, C++ и J#.

Платформа .NET является открытой средой. Это значит, что компиляторы для нее могут поставляться и сторонними разработ-

чиками. К настоящему времени разработаны десятки компиляторов для .NET, например, Ada, COBOL, Delphi, Eiffel, Fortran, Lisp, Oberon, Perl и Python.

Приложение в процессе разработки называется проектом. Проект объединяет все необходимое для создания приложения: файлы, папки, ссылки и прочие ресурсы. Среда Visual Studio.NET позволяет создавать проекты различных типов, например:

- Windows-приложение использует элементы интерфейса Windows, включая формы, кнопки, флажки и пр.;
- консольное приложение выполняет вывод «на консоль», то есть в окно командного процессора;
- библиотека классов объединяет классы, которые предназначены для использования в других приложениях;
- веб-сервис — компонент, методы которого могут вызываться через Интернет.

Консольные приложения наилучшим образом подходят для изучения языка, так как в них не используется множество стандартных объектов, необходимых для создания графического интерфейса.

## **Знакомство со средой**

### **Microsoft Visual Studio 2010 (язык C#).**

**Microsoft Visual Studio 2010** доступна в следующих вариантах:

- **Express** — бесплатная среда разработки, включающая только базовый набор возможностей и библиотек;
- **Professional** — поставка, ориентированная на профессиональное создание программного обеспечения, и командную разра-

ботку, при которой созданием программы одновременно занимаются несколько человек;

- **Premium** – издание, включающее дополнительные инструменты для работы с исходным кодом программ и создания баз данных;
- **Ultimate** – наиболее полное издание Visual Studio, включающие все доступные инструменты для написания, тестирования, отладки и анализа программ, а также дополнительные инструменты для работы с базами данных и проектирования архитектуры ПО.

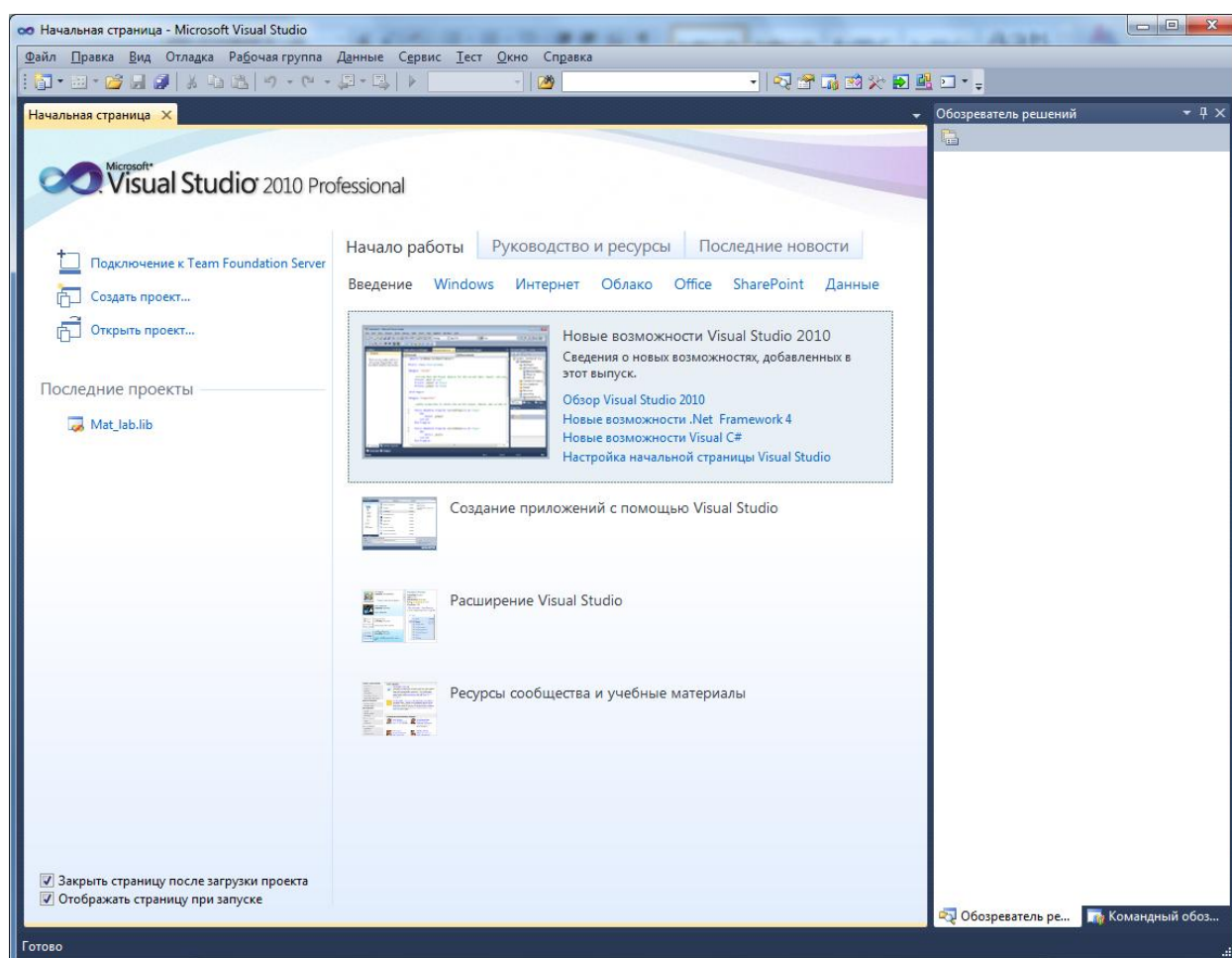


Рис. 0.2. Создание проекта

Отличительной особенностью среды **Microsoft Visual Studio 2010** является то, что она поддерживает работу с несколькими языками программирования и программными платформами. Поэтому, перед тем, как начать создание программы на языке C#, необходимо выполнить несколько подготовительных шагов по созданию проекта и выбора и настройки компилятора языка C# для трансляции исходного кода.

После щелчка на кнопке ОК среда создаст решение и проект с указанным именем. Примерный вид экрана приведен на рис. 2.

В верхней части экрана располагается главное меню (с разделами **Файл, Правка, Вид** и т. д.) и панели инструментов. Панелей инструментов в среде великое множество, и если включить их все, они займут половину экрана. При создании проекта необходимо указать язык C# и тип проекта.

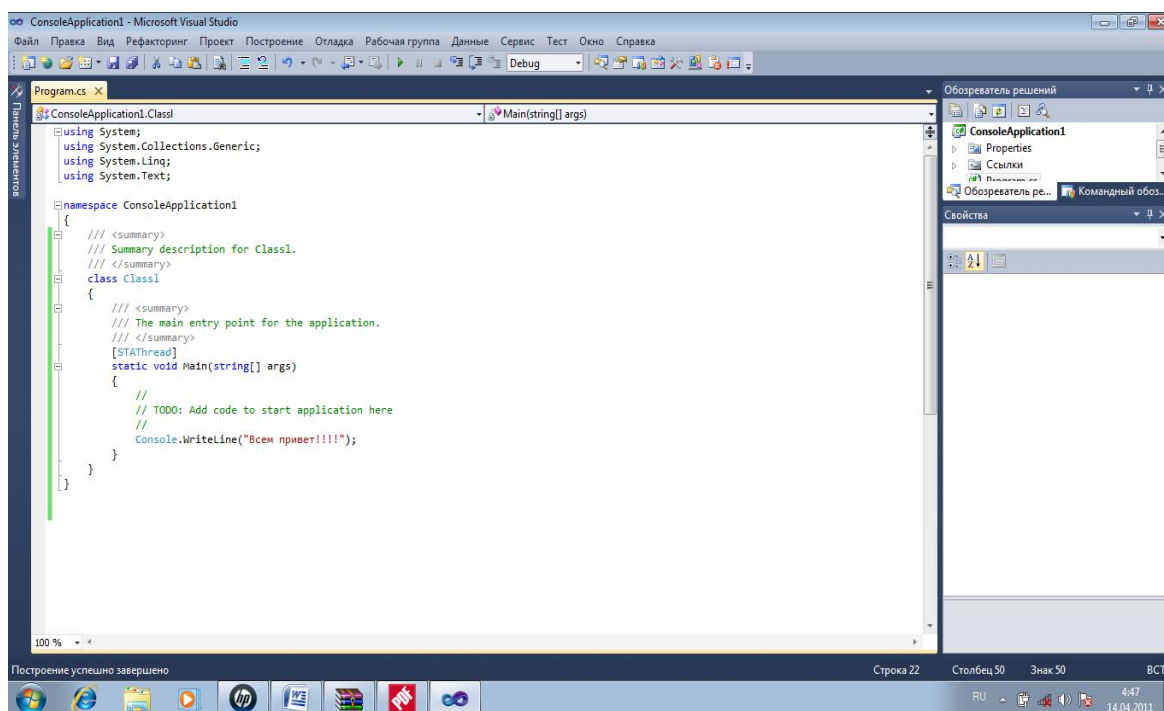


Рис. 0.3. Примерный вид экрана после создания проекта консольного приложения

В верхней правой части экрана располагается окно управления проектом. **Обозреватель решений** (если оно не отображается, следует воспользоваться командой **Вид • Обозреватель решений** главного меню). В окне перечислены все ресурсы, входящие в проект: ссылки на библиотеку, и информация о сборке и файл с исходным текстом класса (Class1.cs). В этом же окне можно увидеть и другую информацию, если перейти на вкладку **Командный обозреватель**, ярлычок которой находится в верхней части окна. На этой вкладке представлен список всех классов, входящих в приложение, их элементов и предков. С помощью проводника Windows можно увидеть какие файлы создала среда для поддержки проекта. На заданном диске появилась папка с указанным именем, содержащая несколько других файлов и вложенных папок. Среди них – файл проекта (с расширением csproj), файл решения (с расширением sln) и файл с кодом класса (Class1.cs).

Основное пространство экрана занимает окно редактора, в котором располагается текст программы, созданный средой автоматически. Текст представляет собой каркас, в который программист добавляет код по мере необходимости. Ключевые (зарезервированные) слова отображаются синим цветом, комментарии (2 различных типов) – серым и темно-зеленым, остальной текст – черным.

Слева от текста находятся символы структуры: щелкнув на любом квадратике с минусом, можно скрыть соответствующий блок кода. При этом минус превращается в плюс, щелкнув на котором, можно опять вывести блок на экран. Это средство хорошо визуализирует структуру кода и позволяет сфокусировать внимание на нужных фрагментах.

## Заготовка консольной программы

Рассмотрим каждую строку заготовки программы (пример 1). Не надо пытаться сразу понять абсолютно все, что в ней написано. Цель – изучить принципы работы в оболочке, а не досконально разобраться в программе.

### Пример 1. Заготовка консольной программы

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
namespace ConsoleApplication1
{
    /// <summary>
    /// Summary description for Class1.
    /// </summary>
    class Class1
    {
        /// <summary>
        /// The main entry point for the application.
        /// </summary>
        static void Main(string[] args)
        {
            //
            // TODO: Add code to start application
            // here
            //
        }
    }
}
```

```
}  
}
```

Директива `using System` разрешает использовать имена стандартных классов из пространства имен `System` непосредственно (без указания имени пространства). Директивы

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

```
using System.Linq;
```

```
using System.Text;
```

относятся к общему случаю и для создания консольного приложения не нужны. Поэтому в дальнейшем они использоваться не будут.

Ключевое слово `namespace` создает для проекта собственное пространство имен, названное по умолчанию `ConsoleApp1`. Это сделано для того, чтобы можно было давать программным объектам имена, не заботясь о том, что они могут совпасть с именами в других пространствах имен.

Строки, начинающиеся с двух или трех косых черт, являются комментариями и предназначены для документирования текста программы.

C# – объектно-ориентированный язык, поэтому написанная на нем программа представляет собой совокупность взаимодействующих между собой классов.

В нашей заготовке программы всего один класс, которому по умолчанию задано имя `Class`. Описание класса начинается с ключевого слова `class`, за которым следуют его имя и далее в фигурных скобках – список элементов класса (его данных и функций, называемых также методами).



## **ВНИМАНИЕ!!!**

Фигурные скобки являются важным элементом синтаксиса. Каждой открывающей скобке соответствует своя закрывающая, которая обычно располагается ниже по тексту с тем же отступом. Эти скобки ограничивают блок, внутри которого могут располагаться другие блоки, вложенные в него, как матрешки. Блок может применяться в любом месте, где допускается отдельный оператор.

В данном случае внутри класса только один элемент – метод `Main`. Каждое приложение должно содержать метод `Main` – с него начинается выполнение программы. Все методы описываются по единым правилам.

Упрощенный синтаксис метода:

```
[ спецификаторы ] тип имя_метода ( [ параметры ] )  
{  
    тело метода: действия, выполняемые методом  
}
```

Наряду с понятием «метод» часто используется другое – функция-член класса. Метод является частным случаем функции – законченного фрагмента кода, который можно вызвать по имени.

Среда поместила внутрь метода `Main` комментарий:

```
// TODO: Add code to start application here
```

Это означает: «Добавьте сюда свой код, выполняемый при запуске приложения». Добавим после строк комментария (но не в той же строке!) строку

```
Console.WriteLine("Всем привет!!!");
```

Здесь `Console` – это имя стандартного класса из пространства имен `System`. Его метод `WriteLine` выводит на экран заданный в ка-

вычках текст. Как видите, для обращения к методу класса используется конструкция

`имя_класса.имя_метода.`

Если вы не сделали ошибку в первом же слове, то сразу после ввода с клавиатуры следом за словом Console символа точки среда выведет подсказку, содержащую список всех доступных элементов класса Console. Выбор нужного имени выполняется либо мышью, либо клавишами управления курсором, либо вводом одного или нескольких начальных символов имени. При нажатии клавиши **Enter** выбранное имя появляется в тексте программы.

Не стоит пренебрегать возможностью автоматического ввода — это убережет от опечаток и сэкономит время. Если подсказка не появляется, это свидетельствует об ошибке в имени или в месте расположения в программе вводимого текста.

Программа должна приобрести вид, приведенный в Примере 2. Обратите внимание на то, что после внесения изменений около имени файла на ярлычке в верхней части окна редактора появился символ \* — это означает, что текст, сохраненный на диске, и текст, представленный в окне редактора, не совпадают. Для сохранения файла воспользуйтесь командой **Файл • Сохранить** главного меню или кнопкой **Save** на панели инструментов (текстовый курсор должен при этом находиться в окне редактора). Впрочем, при выходе из программы среда сохранит исходный текст самостоятельно.

### Запуск программы

Самый простой способ запустить программу — нажать клавишу **F5** (или выбрать в меню команду **Отладка • Начать отладку**). Если программа написана без ошибок, то фраза **Всем привет!!!**

промелькнет перед вашими глазами в консольном окне, которое незамедлительно закроется. Это хороший результат, но для того чтобы пронаблюдать его спокойно, следует воспользоваться клавишами **Ctrl+F5**.

После внесения изменений компилятор может обнаружить в тексте программы синтаксические ошибки. Об этом он сообщает в окне, расположенном в нижней части экрана.

## ПРАВИЛА ВЫПОЛНЕНИЯ РАБОТ

При выполнении лабораторных работ студент обязан.

1. Заранее (дома) подготовиться к лабораторной работе. Для этого необходимо:

- изучить теоретическую часть к лабораторной работе, изложенную в методических указаниях (целесообразно использовать лекции и указанную в них литературу);
- выполнить задание своего варианта, изложенное в методических указаниях;
- оформить отчет по лабораторной работе по следующим правилам.

### Правила оформления отчёта.

Отчёт оформляется на листах формата А4, **строго рукописно** и включает в себя:

- титульный лист, с указанием учебного заведения ( МГТУ «МАМИ» ), кафедры ( кафедра: «Информационные системы и дистанционные технологии»), номера лабораторной работы, её названия, варианта, ФИО и группы студента, ФИО преподавателя и настоящего года;
- теоретическую часть (краткий конспект, минимум две страницы);
- текст задания (строго по тексту методических указаний);
- выполненное задание. Это может быть текст, таблицы, блок-схема, программа, прочее;
- исходные данные, данные для тестирования программы;

МГТУ « МАМИ»

Кафедра: «Информационные  
системы и дистанционные  
технологии»

Лабораторная работа № ?

Название  
(из методических указаний)  
Вариант (выдает преподаватель)

Студент  
группа  
ФИО

Преподаватель  
ФИО

Москва  
2011

- место, оставленное для записи результатов, полученных на занятии с помощью ПК.

2. Представить подготовленный отчёт преподавателю.
3. Получить у преподавателя доступ к ПК и выполнить на нём своё задание.
4. Полученные на ПК результаты показать преподавателю, после чего записать их в отчет (можно в печатном виде).
5. Защитить лабораторную работу.

Защита лабораторной работы включает в себя выполнение контрольного задания преподавателя и ответы на его вопросы по теме лабораторной работы в объёме методических указаний и лекций.

Задание и вопросы преподавателя, решения и ответы студента письменно фиксируются на последнем листе отчёта по лабораторной работе. При защите лабораторной работы студенту разрешается пользоваться конспектом теоретической части его отчёта. После защиты лабораторной работы преподаватель ставит на титульном листе отчёта свою подпись и дату. Только после этого лабораторная работа считается полностью выполненной, и студент может приступать к выполнению следующей.

6. Студент обязан после выполнения всех лабораторных работ сброшюровать все лабораторные работы, сделав для них общий титульный лист, аналогично представленному выше, с названием: Отчёт по лабораторным работам. Дисциплина «Информатика». На титульном листе преподаватель должен сделать запись о допуске студента к экзамену. В таком виде студент должен представить отчёт лектору на экзамене. В противном случае студент к экзамену не допускается.

# ЛАБОРАТОРНАЯ РАБОТА № 1.

## Приложения C# для расчетов по формулам, консольный ввод-вывод.

### 1. Краткие теоретические сведения

#### Типы данных.

Язык C# имеет набор встроенных типов, которые рассматриваются как псевдонимы типов в пространстве имен System. Например, тип `string` – это псевдоним типа `System.String`, а тип `int` – псевдоним типа `System.Int32`. Все встроенные типы подразделены на группы: целочисленные типы; вещественные типы; логиче-

Таблица 1.1

Тип данных	Ключевое слово	Псевдоним класса библиотеки NET	Описание	Размер (бит)
логический	<code>bool</code>	<code>System.Boolean</code>	-	-
целый	<code>Int</code>	<code>System.Int32</code>	со знаком	32
	<code>short</code>	<code>System.Int16</code>	со знаком	16
	<code>byte</code>	<code>System.Byte</code>	без знака	8
	<code>sbyte</code>	<code>System.SByte</code>	со знаком	8
	<code>long</code>	<code>System.Int64</code>	со знаком	64
вещественный	<code>float</code>	<code>System.Single</code>	7 цифр	32
	<code>double</code>	<code>System.Double</code>	15 цифр	64
Строковый символьный	<code>string</code>	<code>System.String</code>	строка	-
	<code>char</code>	<code>System.Char</code>	символов Unicode	16
Любой тип	<code>object</code>	<code>System.Object</code>	объектный	-

ский тип; символьные типы; объектный тип (object). Описание типов приведено в таблице 1.1.

Иерархия классов NET Framework имеет один общий корень – класс System.Object. Все типы разделяются на две категории: размерные типы и ссылочные типы.

При создании переменной размерного типа под нее в стеке выделяется определенный объем памяти, соответствующий типу этой переменной. При передаче такой переменной в качестве параметра выполняется передача значения, а не ссылки на него. Значение размерного типа не может быть равным null. К размерным типам, например, относятся целочисленные и вещественные типы, структуры.

При создании переменной ссылочного типа память под созданный объект выделяется в другой области памяти, называемой кучей. Ссылка всегда указывает на объект заданного типа.

### **Структура приложения на языке C#.**

Проектом называется совокупность файлов, содержащих информацию об установках, конфигурации, ресурсах проекта, а также файлов исходного кода и заголовочных файлов.

Интегрированная среда проектирования Visual Studio позволяет для создания проектов на разных языках программирования использовать различные инструментальные средства проектирования (например, Microsoft Visual Basic, Microsoft Visual C#).

Любое приложение на языке C#, разрабатываемое в среде проектирования Visual Studio, реализуется как отдельный проект. Приложение на языке C# может состоять из нескольких модулей. Каждый модуль C# может содержать код нескольких классов (при соз-

дании приложения в среде Visual Studio.NET каждый класс C# автоматически помещается в отдельный модуль – файл с расширением CS).

Для консольного приложения один из классов, реализуемых модулем, должен содержать метод `Main`. В языке C# нет аппарата заголовочных файлов, используемого в языке C++, поэтому код модуля должен содержать как объявление, так и реализацию класса. По умолчанию весь код класса, представляющего консольное приложение, заключается в одно пространство имен, одноименное с именем приложения.

Точкой входа в программу на языке C# является метод `Main`. Этот метод может записываться как без параметров, так и с одним параметром типа `string` – указателем на массив строк, который содержит значения параметров, введенных при запуске программы. В отличие от списка параметров, задаваемых при запуске C-приложения, список параметров C#-приложения не содержит в качестве первого параметра имя самого приложения. Код метода указывается внутри фигурных скобок:

```
static void Main(string[] args)
{
}
```

Ключевое слово `static` определяет, что метод `Main` является статическим методом, вызываемым без создания экземпляра объекта типа класса, в котором этот метод определен. Метод, не возвращающий никакого значения, указывается с ключевым словом `void`. Однако метод `Main` может возвращать значение типа `int`.

**Пример 1.** Вывод сообщения на консоль.

```
static void Main()
```



```
{
    Console.WriteLine("Ура! \nСегодня
    \"Информатика\"!!!");
}
```



**Замечание.** Для отладки можно использовать команду меню **Debug\Start Without Debugging**. На экране появится окно с результатом исполнения. Обратите внимание на надпись в конце программы: **Press any key to continue**, которая не была предусмотрена. При нажатии любой клавиши окно закрывается. Это результат срабатывания встроенной разработчиками компилятора функции «остановки экрана» для того, чтобы можно было бы сколько угодно долго его рассматривать.

Можно использовать команду **Debug\Start Debugging**, но тогда окно закроется и мы не сможем рассмотреть искомый результат. Для того чтобы обойти это неудобство, следует при разработке программы предусмотреть собственную остановку экрана. Для этого используется команда `Console.Read()`;

## Константы

Это неизменяемые в процессе выполнения программы величины.

**Целые константы** – наиболее распространенный тип `int`. Это целое число, которое может быть отрицательным, положительным или нулем `-12, 5, 0` (все целые со знаком 32 бита). Их можно за-

писывать с суффиксом `-12L` (длинное целое 64 бита), `5u` (целое без знака 8 бит)

**Вещественные константы с фиксированной точкой.** При записи константы типа `float` (32 бита) необходимо, чтобы за значением шел суффикс символ `f` или `F`: `1.2`, `-1.234`, при записи константы типа `double` (64 бита) можно записать суффикс «`d`» или «`D`», но это не является обязательным условием: `1234.5678`, `12.3d`. Дробная часть отделяется от целой части точкой.

**Вещественные константы с плавающей точкой.** При записи константы типа `float` (32 бита) необходимо, чтобы за значением шел суффикс символ `f` или `F`: `1.2E-3f` (число `0.0012`), при записи константы типа `double` (64 бита) `-1.34E5` (число `-134000`) наличие суффикса не требуется.

**Символьные константы.** Символьная константа `char` может представлять собой 16-битный символ `Unicode` («`a`») или управляющие символы (возврат каретки («`\r`»), перевод строки («`\n`»), горизонтальную табуляцию («`\t`»), и другие), заключенный в апострофы.

**Строковые константы** – это последовательность символов, заключенная в кавычки, или константы `string`. Строка, состоящая из символов, например `"Ура! \nСегодня\"Информатика\"!!!"`

**Логическая константа.** Задается одним из двух значений `true` («истина») или `false` («ложь»). Используется в `C#` в логических выражениях, операторах условного перехода.

**Именованные константы.** Применяются для того, чтобы вместо значений констант, использовать в программе их имена, например константа `p` вещественная одинарной точности

```
const float p = 3.14159f
```

## Переменные

Переменная – именованная область памяти, для хранения данных определенного типа. При выполнении программы значение переменной величины можно изменять. Все переменные должны быть описаны явно, при описании переменной задается ее значение и тип. При объявлении переменной может быть задано начальное значение.

Имя переменной может содержать буквы, цифры и символ подчеркивания. Прописные и строчные буквы различаются. Например, переменные Long, LONG, l ong – три разных переменные.

Имя переменной может начинаться с буквы или знака подчеркивания, но не цифры. Имя переменной не должно совпадать с ключевыми словами. Не рекомендуется начинать имя с двух подчеркиваний (такие имена зарезервированы для служебного использования).

Правильные имена переменных: MaxLen, i MaxLen, Max\_Len

Неправильные имена переменных: 2Len, Le#

Примеры описания переменных:

```
int a = -14;           // числовая целая 32 бита
float c = -0.00151f;    // числовая вещественная 32
                        // бита
double i = 1234.56789;  // числовая вещественная 64
                        // бита
bool l = false;         // логическая 16 бит
string name = "Petrov"; // строковая
```

Выражение – состоит из одного или более операндов (которые могут быть переменными, константами, функциями или символьными значениями), знаков операций и круглых скобок.

Примеры выражений:

$2 * 2 + 1$  полученное значение 5

$1 / 2 - 3$  полученное значение -3

$1. / 2 - 3$  полученное значение -2.5

Присвоение значения переменной представляет оператор присваивания (знаки основных операций приведены в таблице 1.2):

$y = 2 * x * x + 3 * x - 1.$

В этом примере сначала производятся вычисления правой части оператора присваивания « = », а затем полученное значение присваивается переменной *y*. Для текстовых данных выражение можно записать в следующем виде:

`string kaf = "Кафедра" + "ИСидТ";`

В этом примере строки по правую сторону от оператора присваивания объединяются, чтобы получить строку “Кафедра + ИСидТ”, которая затем присваивается переменной *kaf*.

Таблица 1.2 Знаки операций

Знак операции	Название
+	Сложение
-	Вычитание
*	Умножение
/	Деление
%	Остаток от деления

Если в арифметических выражениях используются целые числа, то результатом вычислений будет целое число, и любой остаток от деления будет отброшен. Для получения остатка можно использовать соответствующую операцию %, например `10 % 3` возвращает остаток от целочисленного деления, равный 1.

Когда в арифметических выражениях используются числа с плавающей точкой, то результатом деления `10f / 3f` будет число 3,333333.

## Математические функции

C# содержит большое количество встроенных математических функций, которые реализованы в классе `Math` пространства имен `System`.

Рассмотрим краткое описание некоторых математических функций, подробнее с ними можно познакомиться в справочной системе VS или технической документации. Особое внимание следует обратить на типы операндов и результатов, т. к. каждая функция может иметь несколько перегруженных версий.

**Замечание.** Использование нескольких функций с одним и тем же именем, но с различными типами параметров, называется перегрузкой функции. Например, функция `Math.Abs()`, вычисляющая модуль числа, имеет 7 перегруженных версий: `double Math.Abs(double x)`, `float Math.Abs(float x)`, `int Math.Abs(int x)`, и т. д. (таблица 1.3)

Таблица 1.3 Математические функции

№	Название	Описание
1.	<code>Math.Abs(выражение)</code>	Модуль

2.	Math. Ceiling( <i>выражение</i> )	Округление до большего целого
3.	Math. Cos( <i>выражение</i> )	Косинус
4.	Math. E	Число e
5.	Math. Exp( <i>выражение</i> )	Экспонента
6.	Math. Floor( <i>выражение</i> )	Округление до меньшего целого
7.	Math. Log( <i>выражение</i> )	Натуральный логарифм
8.	Math. Log10( <i>выражение</i> )	Десятичный логарифм
9.	Math. Max( <i>выражение1</i> , <i>выражение2</i> )	Максимум из двух значений
10.	Math. Min( <i>выражение1</i> , <i>выражение2</i> )	Минимум из двух значений
11.	Math. PI	Число $\pi$
12.	Math. Pow( <i>выражение1</i> , <i>выражение2</i> )	Возведение в степень
13.	Math. Round( <i>выражение</i> ) Math. Round( <i>выражение</i> , <i>число</i> )	Простое округление Округление до заданного числа цифр
14.	Math. Sign( <i>выражение</i> )	Знак числа
15.	Math. Sin( <i>выражение</i> )	Синус
16.	Math. Sqrt( <i>выражение</i> )	Квадратный корень
17.	Math. Tan( <i>выражение</i> )	Тангенс

**Пример 2.** Вычислить значения функции  $Y = \frac{\cos \pi x}{1+x^2}$  при  $x = 2,5$

using System;

using System.Collections.Generic;

```

using System.Linq;
using System.Text;
namespace ConsoleApplication1
{
    class Example2          // начало описания класса
                            // Example2
    {
        static void Main()
        {
            double p = 3.14159;
            double x = 2.5;
            double y = Math.Cos(p * x)/(1 + x*x);
            Console.WriteLine();
            Console.WriteLine(" x = {0} \t y = {1} ",
                              x, y);
        }
    }
}

```

Эта программа выводит следующее окно с результатом:



**Замечание.** Функция `Console.WriteLine()` выводит на экран пустую строку. Это сделано для более комфортной работы

## Организация ввода-вывода данных.

Программа при вводе данных и выводе результатов взаимодействует с внешними устройствами. Совокупность стандартных устройств ввода (клавиатура) и вывода (экран) называется консолью. В языке C# нет операторов ввода и вывода. Вместо них для обмена данными с внешними устройствами используются специальные объекты. В частности, для работы с консолью используется стандартный класс `Console`, определенный в пространстве имен `System`.

### Ввод данных

Для ввода данных обычно используется метод `ReadLine`, реализованный в классе `Console`. Особенностью данного метода является то, что в качестве результата он возвращает строку (`string`).

#### Пример:

```
static void Main()
{
    string s = Console.ReadLine();
    Console.WriteLine(s);
}
```

Для того чтобы получить числовое значение необходимо воспользоваться преобразованием данных.

#### Пример:

```
static void Main()
{
    string s = Console.ReadLine();
    int x = int.Parse(s); // преобразование строки в
                        // число
    Console.WriteLine(x);
}
```

Или сокращенный вариант:



```
static void Main()
{
    //преобразование введенной строки в число
    int x = int.Parse(Console.ReadLine());
    Console.WriteLine(x);
}
```

Для преобразования строкового представления целого числа в тип `int` мы используем метод `int.Parse()`, который реализован для всех числовых типов данных. Таким образом, если нам потребуется преобразовать строковое представление в вещественное, мы можем воспользоваться методом `float.Parse()` или `double.Parse()`. В случае, если соответствующее преобразование выполнить невозможно, то выполнение программы прерывается и генерируется исключение `System.FormatException` (входная строка имела неверный формат).

### Вывод данных

В приведенных выше примерах мы уже рассматривали метод `WriteLine`, реализованный в классе `Console`, который позволяет организовывать вывод данных на экран. Однако существует несколько способов применения данного метода (таблица 1.4):

Таблица 1.4. Способы вывода

<code>Console.WriteLine(x);</code>	на экран выводится значение идентификатора <code>x</code>
<code>Console.WriteLine("x=" + x + "y=" + y);</code>	на экран выводится строка, образованная последовательным слиянием строки <code>"x="</code> , зна-

	чения $x$ , строки " $y$ =" и значения $y$
<code>Console.WriteLine("x={0} y={1}", x, y);</code>	на экран выводится строка, формат которой задан первым аргументом метода, при этом вместо параметра <code>{0}</code> выводится значение $x$ , а вместо <code>{1}</code> – значение

Если использовать при выводе вместо метода `WriteLine` метод `Write`, вывод будет выполняться без перевода строки.

### **Использование управляющих последовательностей.**

Управляющей последовательностью называют определенный символ, предваряемый обратной косой чертой. Данная совокупность символов интерпретируется как одиночный символ и используется для представления кодов символов, не имеющих графического обозначения (например, символа перевода курсора на новую строку) или символов, имеющих специальное обозначение в символьных и строковых константах (например, апостроф). Рассмотрим управляющие символы (таблица 1.5):

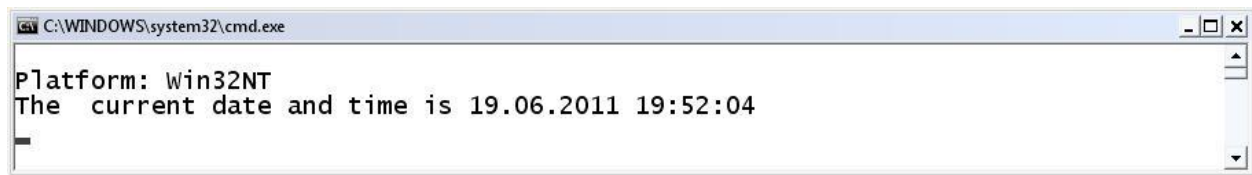
Таблица 1.5. Управляющие символы

Вид	Наименование	Вид	Наименование
<code>\a</code>	Звуковой сигнал	<code>\t</code>	Горизонтальная табуляция
<code>\b</code>	Возврат на шаг назад	<code>\v</code>	Вертикальная табуляция
<code>\f</code>	Перевод страницы	<code>\\</code>	Обратная косая черта

\n	Перевод строки	\'	Апостроф
\r	Возврат каретки	\"	Кавычки

**Пример 3.** Вывести сообщение о версии установленной операционной системы, текущую дату и время.

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
namespace ConsoleApplication1
{
    class Program
    {
        static void Main(string[] args)
        {
            // вывести версию операционной системы
            OperatingSystem os =
                System.Environment.OSVersion;
            Console.WriteLine("Platform:
{0}", os.Platform);
            System.Console.WriteLine("The current
date and time is " +
                System.DateTime.Now);    // дата и время
            System.Console.ReadLine();
        }
    }
}
```



#### Пример 4. Использование консольного ввода для вычисления

значений функции  $Y = \frac{\cos \pi x}{1+x^2}$

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
namespace Lab0
{
    class Program
    {
        static void Main(string[] args)
        {
            System.Console.WriteLine("The current
            date and time is " +
            System.DateTime.Now);
            double pi = 3.14159;
            Console.WriteLine("Input x =\r");
            double x =
            Convert.ToDouble(Console.ReadLine());
            double y = Math.Cos(pi * x)/(1 + x*x);
            Console.WriteLine(" x = {0} \t y = {1}
            ", x, y);
        }
    }
}
```

```

        Console.ReadKey();
    }
}
}

```

```

C:\WINDOWS\system32\cmd.exe
The current date and time is 19.06.2011 19:55:53
Input x =
2
x = 2 y = 0,199999999997183

```

## 2. Практическая часть.

**Задание 1.** Напишите процедуру, выводящую сообщение о версии установленной операционной системы, текущей даты и времени (пример 3).

**Задание 2.** Составить процедуру для выполнения расчетов функции, значения задавать в диалоге с использованием метода `Console.ReadLine()` (пример 4) см. таблицу 1.6;.

Таблица 1.6

Вар.	Функция	x	y
1	$A = \sqrt{\ln(\frac{4}{3} + x) + \frac{9}{7}} - e^{-\sin(1,3x-0,7)}$	0,31 2,5	-0,0049
2	$B = (x + \frac{7}{6})^{\frac{4}{3}} + \sin^x + \arcsin(\cos px)$	-0,75 1,2	-0,018
3	$C = 3,7\sqrt{5-x}\cos(3,5-x) - \sqrt[5]{(5-x)^3}$	2,23 3,2	-0,018

4	$D = -e^{-\cos\sqrt{x+\frac{5}{3}}} - 1,7\arctg(\frac{x}{5} - \frac{3}{4})\sin 1,7x$	-0,35 1,5	-1,318
5	$E = 6,3\sin(1,3x - \frac{p}{3}) - x + \sqrt{x + \frac{9}{4}} + (x + \frac{7}{3})^{\frac{2}{3}}$	0,40 1,5	0,016
6	$F = \cos 1,5x - e^{\sin(x+\frac{4}{3})} + \sqrt{x + \frac{7}{6}}$	2,26 1,2	0,235
7	$G = \frac{5}{3} - \arctg\sqrt{2 - \cos 2x} - e^{-\frac{x}{5}}$	2,09 1,7	0,920
8	$H = \sin\ln(x + 2) - \cos(\pi\ln(x + \frac{5}{3})) + \frac{x}{5}$	-0,26 0,25	-0,0049
9	$I = 4\sin(15e^{\frac{x}{8}} + 10,2) - 9\cos e^{-x} + \sqrt{x + \frac{5}{3}}$	-0,61 0,5	-0,012
10	$J = e^{\frac{4x}{5}} + 2\sin(7\ln(x + \frac{5}{3})) - p$	0,97 -0,5	-0,0024
11	$K = 1,3e^{-\frac{x}{2}} + \left \cos(\frac{2\pi x}{3} - 1,4)\right  - \frac{6}{11}$	2,81 1,25	0,253
12	$L = p + \ln\left \frac{4}{7} - \frac{\sin \arctg x}{2}\right $	2,03 1,7	1,043
13	$M = e^{-\frac{x}{p}} + \frac{4}{3}\arcsin \cos x$	1,97 0,7	0,0017
14	$F = \cos 1,5x - e^{\sin(x + \frac{5}{3})} + \sqrt{x + \frac{7}{6}}$	0,96 1,23	-0,528

15	$O = \arccos \sin(3x + 1,3) - x e^{\arctg x} + 0,7$	1,32 -0,5	0,307
16	$P = 1,3x - 2,5 \sin(5\sqrt{\frac{4}{3} + \arctg x} - 0,7)$	-0,71 0,7	0,0252
17	$Q = e^{-\frac{x}{2}} \cos(2x - 0,3) + \frac{x^2}{2,7 + x}$	-0,73 1,53	-4,197
18	$R = \sqrt{e^{\frac{x}{2}} - 0,1} - x \cos(3x - 1,5)$	2,15 1,2	-1,485
19	$S = 5 \sin(x - 0,3) - \sqrt{2 + e^{-x} - 0,1x^2}$	0,62 1,1	-0,0082
20	$T = 20,7 + (\sin^2(1,2x) - \arccos \frac{x}{8}) \cdot e^{1,5x}$	2,07 1,35	-0,1699
21	$U = 1 - \sin \frac{2x}{3} \cdot \cos \frac{x}{3} - \ln(\frac{5}{3} + x)/1,5$	0,69 1,15	0,0038
22	$V = e^{-\sin \frac{2x}{5}} - e^{\cos \frac{x}{2}} + \sqrt{\frac{4}{3} + x}$	1,28 0,23	-0,0009
23	$W = 2 \sin \frac{4,5x + 0,2}{4} + \sqrt[5]{(x + \frac{7}{4})^2} \cdot e^{-\frac{x}{4}}$	-0,63 1,35	6,0827
24	$X = \cos(4 \ln(x + \frac{7}{3})) - \ln(4 - \frac{x}{7})/3 - \frac{4}{11}$	1,78 2,3	0,0064
25	$Y = \arcsin \sqrt{\frac{1}{3} + \frac{x}{7}} - 1,5 \sin \ln(x + \frac{4}{3})$	0,23 1,4	-0,0021

26	$Z = \cos \frac{x}{3} - e^{-(x+1)^2} - \frac{4}{9}$	-0,23 0,96	-1,0396
27	$A1 = 8x / (\frac{70}{3} + 7\sqrt{\frac{7}{6} + x}) - e^{-\sin^2(\frac{2x}{3})}$	1,83 -0,5	0,2601
28	$B1 = \frac{15}{9} - (\sqrt{\frac{1}{3} + \frac{x}{5}} + \sqrt[3]{ x }) / e^{-\frac{x}{3}}$	0,47 1,2	-0,0073
29	$C1 = \ln(\frac{5}{3} + x) - \sin \sqrt{\frac{7}{3} + x} + \frac{1}{7}$	0,66 -0,5	-0,0001
30	$D1 = \frac{1}{3}e^{-\cos^2 x} + \sqrt{x + \frac{9}{7}} / (1 + \ln(x + 3)) - \frac{4}{5}$	0,71 1,4	-0,0012



## **ЛАБОРАТОРНАЯ РАБОТА № 2**

### **Ввод-вывод информации, с использованием файлов.**

#### **Форматирование значений данных.**

### **1. Краткие теоретические сведения**

Вывод, производимый методами `System.Console.WriteLine()` и `System.Console.WriteLine()`, можно форматировать. Форматирование позволяет указывать формат целых чисел, чисел с плавающей точкой и других типов данных.

#### **Управление форматом числовых данных**

Пусть в программе определена переменная типа `int` с именем `value`:

```
int value = 250;
```

До этого момента переменные выводились следующим образом:

```
System.Console.WriteLine("value =" + value);
```

Результат вывода: `value = 250`

Можно вывести значение `value`, используя требуемое число позиций (например 5):

```
System.Console.WriteLine("value = {0, 5}", value);
```

Первое число в фигурных скобках означает номер переменной – это 0, что соответствует первой переменной `value` в списке метода `System.Console.WriteLine()`. Второе число в фигурных скобках означает количество позиций, отведенное для отображения переменной. В данном примере оно равно 5. При выводе переменной длина ее представления будет дополнена пробелами сле-

ва. Если количество позиций меньше чем число знаков переменной, то оно будет выведено без форматирования.

Можно задать форматирование для вывода каждой переменной:

```
int a = -12;  
int b = 20;  
System.Console.WriteLine("a = {0, 4}, b = {1, 3}",  
a, b);
```

Результат вывода: a = -12, b = 20

Форматированный вывод чисел с плавающей точкой немного более сложный. Предположим, определена переменная типа double с именем myDouble: double myDouble = 1234.56789;

Следующий пример выводит значения myDouble, отведя под него десять знакомест, и округлив его до трех цифр после запятой:

```
System.Console.WriteLine("myDouble = {0, 10: f3}",  
myDouble);
```

Символы f3 в этом примере означают, что значение выводится как число с плавающей точкой (символ f) , в дробной части будет выведено три цифры.

Точно такое же форматирование можно применять для типов float и decimal. Например:

```
float myFloat = 1234.56789f;  
System.Console.WriteLine("myFloat = {0, 10: f3}",  
myFloat);  
decimal myDecimal = 1234.56789m;  
System.Console.WriteLine("myDecimal = {0, 10: f3}",  
myDecimal);
```

Результат вывода:

```
myFloat = 1234.568;
```

```
myDecimal 1234.568;
```

В списке аргументов методов `WriteLine` или `Write` задается строка вида `{n, w: спецификатор k}` – где `n` определяет номер идентификатора из списка аргументов метода `WriteLine`, *спецификатор* – определяет формат данных, `w` – целая константа без знака, задает количество символов (длину поля), а `k` – количество позиций для дробной части значения идентификатора.

Для каждого типа данных существует своя форма представления. Данные сведены в таблицу 2.1.

Таблица 2.1

Тип данных		Форма
Числовые	Целые	W
	Вещественные с фиксированной точностью	W: Fk
	Вещественные в экспоненциальном формате	W: Ek
Логические		W
Символьные		W

Символы форматирования F, E (другие символы форматирования приведены в табл. 2.2) – определяют тип и характеристики объектов ввода-вывода. Параметр `w` – целая константа без знака, задает количество символов (длину поля), отводимых для ввода-

вывода объекта. Параметр *k* – целая константа без знака определяет для числовых данных:

- количество позиций, для цифр в дробной части числа (форма F);
- количество позиций для цифр, в дробной части мантииссы числа (форма E или G).

В качестве спецификаторов могут использоваться следующие значения:

Таблица 2.2

Символ	Формат	Значение
С или с	Денежный. По умолчанию ставит знак р. Изменить его можно с помощью объекта <code>NumberFormatInfo</code>	Задается количество десятичных разрядов.
D или d	Целочисленный (используется только с целыми числами)	Задается минимальное количество цифр. При необходимости результат дополняется начальными нулями
E или e	Экспоненциальное представление чисел	Задается количество символов после запятой. По умолчанию используется 6
F или f	Представление чисел с фиксированной точкой	Задается количество символов после запятой
G или g	Общий формат (или экспоненциальный, или с фиксированной точкой)	Задается количество символов после запятой. По умолчанию выводится целая часть

N или n	Стандартное форматирование с использованием запятых и пробелов в качестве разделителей между разрядами	Задается количество символов после запятой. По умолчанию – 2, если число целое, то ставятся нули
X или x	Шестнадцатеричный формат	
P или p	Процентный	

**Пример 1.** Форматированный вывод данных различного типа.

```
public static void Main()
{
    int a = -14;
    float c = -0.00151f;
    double i = 1234.56789;
    bool l=false;
    string name="Petrov";
    System.Console.WriteLine("name = {0, 6}, l =
{1, 4}", name, l);
    System.Console.WriteLine("a ={0, 4}, c =
{1, 10: f5}, i = {1, 20: e8}", a, c, i);
    System.Console.WriteLine(" ");
    System.Console.WriteLine("Для выхода нажмите
на Enter");
    System.Console.ReadLine();
}
```

```
file:///c:/Example/ConsoleApplication6/ConsoleApplication6/bin/Debug/ConsoleApplication6.EXE

Input фамилию
Petrov
Input a
-14
Input c
-0,00151
Input i
1234,56789
Input l
false
Результаты форматирования
name = Petrov, l = False
a = -14, c = -0,00151, i = 1,23456789e+003

Для выхода нажмите на Enter
_
```

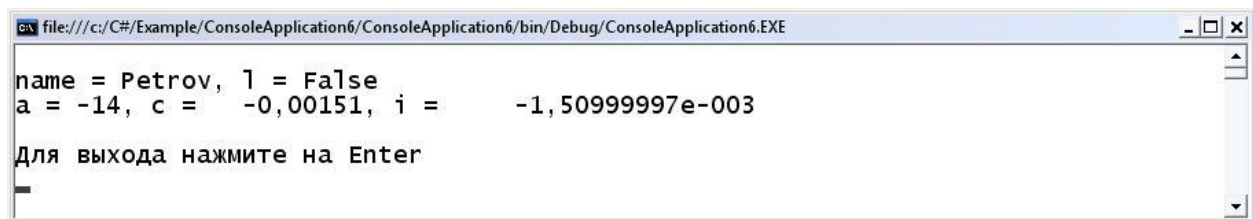
**Пример 2.** Ввод в диалоге и форматированный вывод данных различного типа.

```
public static void Main()
{
    int a;        // = -14;
    float c;      // = -0.00151f;
    double i;     // = 1234.56789;
    bool l;       // = false;
    string name;  //="Petrov";
    Console.WriteLine("Input фамилию ");
    name = Console.ReadLine();
    Console.WriteLine("Input a");
    a = Convert.ToInt32(Console.ReadLine());
    Console.WriteLine("Input c");
    c = Convert.ToSingle(Console.ReadLine());
    Console.WriteLine("Input i");
    i = Convert.ToDouble(Console.ReadLine());
    Console.WriteLine("Input l");
    l = Convert.ToBoolean(Console.ReadLine());
```

```

System.Console.WriteLine(" Результаты
форматирования \n name = {0, 6}, l = {1, 4}",
name, l);
System.Console.WriteLine("a ={0, 4}, c =
{1, 10: f5}, i = {2, 20: e8}", a, c, i);
System.Console.WriteLine(" ");
System.Console.WriteLine("Для выхода нажмите
на Enter");
System.Console.ReadLine();
}

```



## Организация ввода вывода с использованием файлов

C#-программы выполняют операции ввода-вывода посредством потоков, которые построены на иерархии классов. Поток (stream) – это абстракция, которая генерирует и принимает данные. С помощью потока можно читать данные из различных источников (клавиатура, файл) и записывать в различные источники (принтер, экран, файл). Несмотря на то, что потоки связываются с различными физическими устройствами, характер поведения всех потоков одинаков. Поэтому классы и методы ввода-вывода можно применить ко многим типам устройств.

На самом низком уровне иерархии потоков ввода-вывода находятся потоки, оперирующие байтами. Это объясняется тем, что многие устройства при выполнении операций ввода-вывода ориен-

тированы на байты. Однако для человека привычнее оперировать символами, поэтому разработаны символьные потоки, которые фактически представляют собой оболочки, выполняющие преобразование байтовых потоков в символьные и наоборот. Кроме этого, реализованы потоки для работы с `int`-, `double`-, `short`- значениями, которые также представляют оболочку для байтовых потоков, но работают не с самими значениями, а с их внутренним представлением в виде двоичных кодов.

Центральную часть потоковой C#-системы занимает класс `Stream` пространства имен `System.IO`. Класс `Stream` представляет байтовый поток и является базовым для всех остальных потоковых классов. Чтобы создать символьный поток нужно поместить объект класса `Stream` (например, `FileStream`) "внутри" объекта класса `StreamWriter` или объекта класса `StreamReader`. В этом случае байтовый поток будет автоматически преобразовываться в символьный.

**Класс `StreamWriter`** предназначен для организации выходного символьного потока. Этот класс содержит несколько конструкторов. Так, например, создать экземпляр класса `StreamWriter` можно следующим образом:

```
StreamWriter fileOut = new StreamWriter(new  
FileStream("text.txt", FileMode.Create,  
FileAccess.Write));
```

Эта версия конструктора позволяет ограничить доступ только чтением или только записью:

```
FileStream(string filename, FileMode mode,  
FileAccess how)
```

где:



1. параметры `filename` и `mode` имеют то же назначение, что и в предыдущей версии конструктора;
2. параметр `how`, определяет способ доступа к файлу и может принимать одно из значений, определенных перечислением `FileAccess`:

`FileAccess.Read` - только чтение;

`FileAccess.Write` - только запись;

`FileAccess.ReadWrite` - и чтение, и запись.

Другой вид конструктора позволяет открыть поток сразу через обращения к файлу:

`StreamWriter(string name),`

где параметр `name` определяет имя открываемого файла.

Например, обратиться к данному конструктору можно следующим образом:

```
StreamWriter fileOut = new
```

```
StreamWriter("c:\temp\t.txt");
```

И еще один вариант конструктора `StreamWriter`:

`StreamWriter(string name, bool appendFlag),`

где параметр `name` определяет имя открываемого файла; параметр `appendFlag` может принимать значение `true` – если нужно добавлять данные в конец файла, или `false` – если файл необходимо перезаписать.

Например:

```
StreamWriter fileOut=new StreamWriter("t.txt", true);
```

Теперь для записи данных в поток `fileOut` можно обратиться к методу `WriteLine`. Это можно сделать следующим образом:

```
fileOut.WriteLine("test");
```

В данном случае в конец файла `t.txt` будет дописано слово `test`.

**Класс `StreamReader`** предназначен для организации входного символьного потока. Один из его конструкторов выглядит следующим образом:

```
StreamReader(Stream stream),
```

где параметр `stream` определяет имя уже открытого байтового потока. Этот конструктор генерирует исключение типа `ArgumentException`, если поток `stream` не открыт для ввода.

Например, создать экземпляр класса `StreamWriter` можно следующим образом:

```
StreamReader fileIn = new StreamReader(new  
FileStream("text.txt", FileMode.Open,  
FileAccess.Read));
```

Как и в случае с классом `StreamWriter` у класса `StreamReader` есть и другой вид конструктора, который позволяет открыть файл напрямую:

```
StreamReader (string name);
```

где параметр `name` определяет имя открываемого файла.

Обратиться к данному конструктору можно следующим образом:

```
StreamReader fileIn=new StreamReader  
("z:\temp\t.txt");
```

В C# символы реализуются кодировкой `Unicode`. Для того, чтобы можно было обрабатывать текстовые файлы, содержащие русский символы, созданные, например, в Блокноте, рекомендуется вызывать следующий вид конструктора `StreamReader`:

```
StreamReader fileIn=new StreamReader  
("z:\temp\t.txt", Encoding.GetEncoding(1251));
```

Параметр `Encoding.GetEncoding(1251)` говорит о том, что будет выполняться преобразование из кода `Windows-1251` (одна из

модификаций кода ASCII, содержащая русские символы) в Unicode. Encoding.GetEncoding(1251) реализован в пространстве имен System.Text.

Теперь для чтения данных из потока `FileIn` можно воспользоваться методом `ReadLine`. При этом если будет достигнут конец файла, то метод `ReadLine` вернет значение `null`.

По завершении работы с файлом его необходимо закрыть. Для этого достаточно вызвать метод `Close()`. При закрытии файла освобождаются системные ресурсы, ранее выделенные для этого файла, что дает возможность использовать их для работы с другими файлами.

Рассмотрим пример, в котором данные из одного файла считываются программой расчета функции и результаты помещаются в другой файл в заданной форме с использованием классов `StreamWriter` и `StreamReader`.

**Пример 3.** Ввод данных из файла и форматированный вывод данных различного типа в файл.

```
static void Main()
{
    string s;
    double x, y;
    StreamWriter f = new StreamWriter("out.txt");
    StreamReader f1 = new StreamReader("in.txt");
    f.WriteLine("        Таблица значений\n");
    metka: s = f1.ReadLine();
    if (s == null) goto metka1;
    x = Convert.ToDouble(s);
```

```

y = Math.Sqrt(x * x / (2 + Math.Exp(4 *
Math.Log(x))));
f.WriteLine(" аргумент x = {0:F3} функция y =
{1:e3} \n", x, y);
goto metka;
metka1: f.WriteLine("    Составил  Петров Иван
{0} \n", s);
f.Close();
f1.Close();
}

```

#### **Исходные данные файл in.txt**

**0,11**  
**0,5**  
**1**

#### **Результаты расчетов файл out.txt**

##### **Таблица значений**

**аргумент x = 0,110 функция y = 7,778e-002**

**аргумент x = 0,500 функция y = 3,482e-001**

**аргумент x = 1,000 функция y = 5,774e-001**

**Составил Петров Иван**

## **2. Практическая часть**

1) Составить программу для ввода в диалоге значений переменных A, I, C, L, Name и форматного вывода на экран монитора

введенных переменных (значения вводимых переменных даны в таблице 2.3).

2) Составить программу для вычисления и печати значений функции из таблицы 2.4. Вычислить 8 значений функции на заданном интервале. Исходные данные задать в файле LAB2. TXT. Результат поместить в файл вывода с именем LAB2. RES в заданной форме (таблица 2.5) .

### Варианты задания

Таблица 2.3

Вариант	A	I	C	L	N
1	-14	$-10^4$	-0,00151	ложь	Фамилия
2	99,35	72	1995	истина	Имя
3	0,086	-19	4,025	ложь	Отчество
4	34	-6124	$3,2 \times 10^5$	истина	Фамилия
5	5,008	229	0,019	ложь	Имя
6	$3,5 \times 10^{-4}$	1989	-380,08	истина	Отчество
7	0,095	-1	1996	ложь	Фамилия
8	1,0074	$10^2$	107,7	истина	Имя
9	993,285	112000	$2,3 \times 10^{-4}$	ложь	Отчество
10	-2,1	444	$10^3$	истина	Фамилия
11	3,125	6006006	-13,24	ложь	Имя
12	-45,077	30	$25 \times 10^{12}$	истина	отчество
13	12,97	1002	-999,7	ложь	фамилия
14	-0,09	2004	399,44	ложь	имя
15	-142	$-10^4$	-0,00151	истина	отчество
16	9,35	-5072	19,95	ложь	фамилия
17	0,86	-19726	4,025	истина	имя

18	34	-6	$3,2 \times 10^3$	ложь	отчество
19	5,008	-229	-0,019	истина	фамилия
20	$3,5 \times 10^{-4}$	1989	-380,08	ложь	имя
21	0,095	-12	1996	истина	отчество
22	1,0074	$10^2$	107,7	ложь	фамилия
23	993,285	112000	$2,3 \times 10^{-4}$	истина	имя
24	$-2,1 \times 10^3$	444	$10^{-3}$	ложь	отчество
25	3,125	6007007	-13,24	истина	фамилия
26	-45,07	123	$25 \times 10^{12}$	ложь	имя
27	89,09	1000	999,002	истина	отчество
28	-99,78	11	-1,774	ложь	фамилия
29	7,99	-30077	1000	истина	истина
30	0,124	-100400	-9000	ложь	фамилия

Таблица 2.4

№	Функция	Контрольное значение		Интервал x		Вариант формы вывода
		X*	y*	X <sub>min</sub>	X <sub>max</sub>	
1	$y = \frac{1-x^2}{1+x^4}$	2	-0,176	-3	3	1
2	$y = \frac{\sin x}{x^2 - 1}$	1,57	0,406	-2	2	2
3	$y = \frac{2\pi}{x^2 - \pi}$	3,14	0,935	-2	4	3
4	$y = 4x + \frac{1}{x+1}$	1	4,5	0	2,5	4

5	$y = \frac{\sin^2 x}{x-1}$	1,57	1,75	1,5	5	1
6	$y = \frac{x^2 - 5x + 4}{x^2 + 1}$	2	-0,4	-2	3	2
7	$y = 2 - \frac{e^{2x} + e^{-2x}}{e^2 + e^{-2}}$	1	1	-1	1	3
8	$y = \frac{\sin x}{x^2 + 1}$	1,57	0,299	-2	2	4
9	$y = e^{-x} \sin\left(\frac{\pi x}{2}\right)$	-1	-2,7	0	2,5	1
10	$y = \frac{2\pi}{(x^2 + \pi)}$	0,5	1,9	-3	3	2
11	$y = \ln \pi \sqrt{x^3 + x^2}$	-0,6	0,43	-1	1,5	3
12	$y = x^3 \cos^2(x + 3)$	0,14	0,0027	-2	2	4
13	$y = \sqrt{\frac{1}{2\pi}} e^{-x+1}$	1,5	0,242	0	3	1
14	$y = \sqrt{x} + \sqrt{3-x}$	1	2,4	0	4	2
15	$y = 2 \operatorname{arctg} x + \sin \pi x$	1	1,57	-2	2	3
16	$y = x^2 - 3 \sin x$	1,57	-0,53	-0,5	2	4
17	$y = x (1 + \cos \pi x)$	0,5	0,5	-1,5	1,5	1
18	$y = \sqrt{x} e^{-x}$	1	0,369	0	3	2
19	$y = \operatorname{tg} x - 3x^2$	1,2	-1,75	-1,3	1,3	3
20	$y = e^{-x} \sin^2 x$	1,2	0,262	-2	2	4

21	$y = \ln(x-1) \cos^2 x$	3	1,077	1	4	1
22	$y = x(1 + \cos \pi x)$	-0,5	-0,5	-1,5	1,5	2
23	$y = \sqrt{e}   x^2 - 1  - 2 $	0,25	1,75	-2	2	3
24	$y = \sqrt{\pi} x \arctg x$	1	1,4	-2	3	4
25	$y = x - \sin \pi x$	0,5	-0,5	-2	3	1
26	$y = x^3 - 5x^2 + 4\sqrt{x}$	0,5	0,875	-1	4	2
27	$y = 1 + 2 \cos \pi x - \sin 2x$	0	3	-2	2	3
28	$y = \cos^2 \pi x - 2$	0	-1	-2	2	4
29	$y = 2x - 5x^{\frac{2}{3}}$	-1	-7	-2	3	1
30	$y = 2(x^3 - 9x^2)^{\frac{1}{3}}$	1	-4	-2	5	2

Таблица 2.5

Вариант формы вывода	Форма вывода информации 7890123456789012345678901234 – позиции
1	<p>Таблица значений</p> <p>I-----I</p> <p>I    X        I    Функция        I</p> <p>I-----I</p> <p>I X =...        I    Y =...        I</p> <p>I X =...        I    Y =...        I</p> <p>I-----I</p> <p>Составил: &lt; Ф. И. О. &gt;</p>
2	Таблица



	<p>*****</p> <p>* X = ...                      *        Y = ...                      *</p> <p>*****</p> <p>* X = ...                      *        Y = ...                      *</p> <p>*****</p> <p>Составил: &lt; Ф. И. О. &gt;</p>
3	<p>Таблица значений</p> <p>+-----+</p> <p>+ Аргумент    +    Функция            +</p> <p>+-----+</p> <p>+    X = ...        +    Y = ..                    +</p> <p>+    X = ...        +    Y = ...                    +</p> <p>+            ...        +            ...                    +</p> <p>+-----+</p> <p>Составил: &lt; Ф. И. О. &gt;</p>
4	<p>Получено:</p> <p>для заданной функции <math>Y(\dots) = \dots</math></p> <p>для заданной функции <math>Y(\dots) = \dots</math></p> <p>Составил: &lt; Ф. И. О. &gt;</p>

## **ЛАБОРАТОРНАЯ РАБОТА № 3**

### **Управляющие операторы условного и безусловного переходов.**

#### **Разветвляющиеся программы**

#### **1. Краткие теоретические сведения**

Как известно, из предыдущих лабораторных работ, все программы состоят из последовательности операторов, которые обычно выполняются поочерёдно в том порядке, в каком они записаны в программе. Однако часто возникает необходимость изменить очередность выполнения операторов, т.е. пропустить или наоборот выполнить какую-то группу операторов в зависимости от выполнения или не выполнения некоторых заданных условий. Кроме того иногда необходимо повторить группу операторов несколько раз, то есть организовать цикл. Для выполнения этих задач служат управляющие операторы. Управляющие операторы подразделяются на операторы принятия решения, к ним относятся операторы условного и безусловного переходов, и операторы для организации циклов, которые будут рассмотрены в следующей лабораторной работе.

Одной из важнейших возможностей компьютерного процессора является возможность принятия решения. Под этим выражением подразумевается, что процессор может направить поток выполнения запрограммированных команд по тому или иному пути в зависимости от того истинно или ложно некоторое заданное условие. Любой язык программирования обеспечивает возможность принятия решения. В алгоритмическом языке С#, как и во многих других, основой для такой возможности является оператор услов-

ного перехода  $i f$ , который действует в C# практически также, как и оператор IF в любом другом языке программирования./

### **Оператор условного перехода $i f$ и его конструкции**

Оператор условного перехода  $i f$  (если), как было уже сказано, предназначен для выбора одного из возможных вариантов исполнения операторов программы, поэтому его и называют оператором принятия решения. Существует несколько разновидностей конструкций этого оператора. Рассмотрим их последовательно по мере усложнения.

$i f$  (*условие выбора*)

{

    // Записанные в скобках операторы (оператор)

    // будут выполняться, если *условие выбора* истинно

}

// Записанные далее операторы будут выполняться

// в любом случае, независимо от *условия выбора*.

В качестве *условия выбора* используется значение логического выражения. При выполнении этой конструкции вначале вычисляется значение логического выражения, записанного в скобках. Результат вычисления имеет тип `bool` `en`. Если вычисленное выражение имеет значение `true` (истина), то выполняются операторы в фигурных скобках и все следующие за ними. Если получено значение `false` (ложь) то операторы в фигурных скобках пропускаются и выполняются только операторы следующие за ними.

### **Пример 1.**

```
// Сохраняем наибольшее значение из двух, a и b,  
// в переменной max  
max = b;  
if (a > b)  
{  
    max = a;  
}
```

В данном фрагменте программы изначально предполагается, что наибольшее значение имеет переменная *b*, и она присваивается переменной *max*. Если это не так, то переменной *max* присваивается значение переменной *a*.

Операторы, записываемые в фигурных скобках можно размещать в одной строке, как в следующем примере.

### **Пример 2.**

```
max = b;  
if (a > b){max = a; }
```

При записи в фигурных скобках нескольких операторов в конце каждого из них ставится точка с запятой. Если в фигурных скобках записывается один оператор, то фигурные скобки можно опустить (см. пример 3).

### **Пример 3.**

```
max = b;  
if (a > b) max = a;
```

### Конструкция if-else (если -иначе)

Данную конструкцию целесообразно использовать, когда необходимо задать выполнение одного из двух блоков операторов (или одного из двух операторов), в зависимости от результата проверки *условия выбора*. Конструкция имеет следующий вид записи.

```
if (условия выбора)
{
    // Если условие выбора истинно, то будут выпол-
    // няться оператор или операторы блока 1.
}
else
{
    // В противном случае (иначе)
    // если условие выбора ложно, то будут выполнять-
    // ся оператор или операторы блока 2.
}
// Записанные далее операторы будут выполняться
// в любом случае, независимо от условия выбора.
```

Если результатом проверки *условия выбора* является значение true (истина), то будут выполнены *операторы блока 1*. Далее будет выполняться первый оператор, следующий за последней фигурной скобкой. *Операторы блока 2* выполняться не будут. Если проверка *условия выбора* даст результат false (ложь), то *операторы блока 1* будут пропущены, и будут выполнены *операторы блока 2*. Далее будет выполняться первый

оператор, следующий за последней фигурной скобкой. Каждый из указанных блоков может состоять из одного оператора, тогда фигурные скобки могут быть опущены.

**Пример 4.**

```
if (a > b)
{
    max = a; // Эти операторы будут выполняться,
    min = b; // если условие выбора a > b истинно.
}
else
{
    max = b; // Эти операторы будут выполняться,
    min = a; // если условие выбора a > b ложно.
}
// Записанные далее операторы будут выполняться
// в любом случае, независимо от условия выбора.
```

Если  $a > b$ , то переменной *max* будет присвоено значение *a*, переменной *min* - значение *b*, в противном случае наоборот переменной *max* присваивается значение *b*, а переменной *min* - значение *a*. Рассмотренную конструкцию допускается записывать в одной строке, как в следующем примере 5.

**Пример 5.**

```
if (a > b)
{
```

```

    max = a;
    min = b;
}
else
{
    max = b;
    min = a;
}

```

Если в фигурных скобках записано по одному оператору, то скобки можно опустить, как в примере 6.

#### **Пример 6.**

```

if (a > b) max = a;
else max = b;

```

#### **Вложенные конструкции оператора if**

В том случае, когда определённый *блок операторов* (или один оператор) нужно выполнить после проверки ни одного, а нескольких условий, то используют несколько конструкций оператора if. Операторы if, находящиеся внутри другого оператора if, называются вложенными конструкциями оператора if.

```

if (Условие 1 выбора)
{
    // Если условие 1 выбора истинно будут выполня-
    // ться, записанные в скобках операторы блока 1.
}

```

```

else
// В противном случае будет выполняться
// вложенная конструкция оператора if.
{
    // Начало вложенной конструкции оператора if.
    if (условие 2 выбора)
    {
        // Если условие 2 выбора истинно будут
        // выполняться, записанные здесь в скобках
        // операторы блока 2.
    }
    Else
    {
        // В противном случае,
        // если условие 2 выбора ложно будут вы
        // полняться, записанные здесь в скобках
        // операторы блока 3.
    }
} // Конец вложенной конструкции оператора if

```

Если *условие 1 выбора* истинно, то выполняются *операторы блока 1*, и далее первый оператор, который следует за последней фигурной скобкой, концом вложенной конструкции оператора *if*. В противном случае выполняется вложенная конструкция оператора *if*. Если *условие 2 выбора* вложенного оператора *if* истинно, то выполняются записанные в фигурных скобках *операторы блока 2*, и далее первый оператор, который следует за последней фигурной скобкой, концом вложенной конструкции опе-



ратора `if`. В противном случае, если *условие 2 выбора* ложно выполняются *операторы блока 3*. Рассмотрим пример записи вложенной конструкции оператора `if`.

### Пример 7.

```
. . .  
if (x < -1)  
{ n = 1; }  
else  
{  
    // Начало вложенной конструкции if.  
    if (x > 1)  
    { n = 2; }  
    else  
    { n = 0; }  
    // Конец вложенной конструкции if.  
}  
. . .
```

Допускаются и другие виды записи вложенной конструкции оператора `if`, например запись в одной строке.

### Пример 8.

```
. . .  
if (x < -1)  
{ n = 1; }  
else  
{  
    if (x > 1) { n = 2; }  
    else { n = 0; }  
}
```

```
}
```

```
. . .
```

Поскольку фигурные, операторные скобки являются обязательными только в случае записи в них нескольких операторов, поэтому в данном случае они могут быть опущены.

### **Пример 9.**

```
. . .
```

```
if (x < -1) n = 1;
```

```
else
```

```
if (x > 1) n = 2;
```

```
else n = 0;
```

```
. . .
```

## **Операторы логического сравнения**

Эти операторы называются логическими сравнениями (logical comparisons), поскольку они возвращают результат сравнения в виде значения `true` (истина) или `false` (ложь) имеющие тип `bool`. Для записи и проверки условия равенства двух выражений, в алгоритмическом языке `C#` используется символ `==`. Аналогично: символ `>` используется для проверки условия «больше»; символ `<` для проверки условия «меньше»; `>=` – «больше или равно»; `<=` – «меньше или равно»; `!=` «не равно». Например: `a!=b`, означает, что оператор логического сравнения `!=` возвращает значение `true`, если `a` не равно `b`.

## **Логические операторы**

Для переменных типа `bool` используются специальные составные логические операторы:

$\&$  – конъюнкция (логическое и, and), используется для логического объединения двух выражений;

$|$  – дизъюнкция (логическое или, or), используется, чтобы убедиться в том, что хотя бы одно из выражений true, истинно;

$!$  – отрицание (логическое не, not), возвращает обратное логическое выражение;

$\wedge$  – исключение (логическое исключаящее или), используется для того, чтобы убедиться в том, что одно из двух выражений true, истинно.

Операторы  $\&$ ,  $|$  и  $\wedge$  обычно используются с целыми типами данных, а также могут применяться к типу данных bool.

Кроме того могут применяться операторы  $\&\&$  и  $||$ , которые отличаются от своих односимвольных версий тем, что выполняют ускоренные вычисления. Например в выражении  $a \&\& b$ ,  $b$  вычисляется лишь в том случае, если  $a$  равно true, истинно. В выражении  $a || b$ ,  $b$  вычисляется в том случае, если  $a$  равно false, ложно.

### **Пример 10.**

`if (x > -1 && x < 1)`

В условии оператора `if` записано обычное алгебраическое неравенство  $-1 < x < 1$ .

### **Пример 11.**

`if (x < -1 || x > 1)`

В условии оператора `if` записаны алгебраические неравенства  $x < -1$  либо  $x > 1$ .

## Оператор проверки

Оператор проверки выбирает одно из двух выражений в зависимости от проверки значения логического условия. Его синтаксис:

Имя переменной = (*условие выбора*)?Значение1: значение2

### Пример 12.

```
int value = (x < 25) ? 5: 15
```

В этом примере сначала вычисляется выражение  $x < 25$  являющееся *условием выбора*. Если оно равно true, то переменной `value` будет присвоено значение равное 5, в противном случае – равное 15.

## Оператор безусловного перехода goto

Оператор безусловного перехода `goto` (перейти к) осуществляет переход, без проверки каких-либо условий, к оператору, обозначенному соответствующей меткой. Синтаксис этого оператора выглядит следующим образом:

*метка:*    *оператор*

. . .

`goto`    *метка*

. . .

где *метка* - метка. Это любой допустимый идентификатор C#, который помещается слева от *оператора*, которому надо передать управление выполнением программы и отделяется от него двоеточием. Причём *метка* может ставиться у *оператора* расположенного как до оператора `goto`, так и после него. В случае если оператор `goto` используется самостоятельно, без каких либо конструкций, то первый оператор, следующий за оператором `goto`, дол-

жен иметь свою метку, иначе он не будет выполнен в процессе работы программы. Обычно оператор `goto` используется совместно с оператором условного перехода `if`, и используется в программах редко, т. к. есть более эффективные операторы.

### Конструкция `switch` (переключатель)

Этот оператор позволяет сделать выбор среди нескольких альтернативных вариантов дальнейшего выполнения программы. Несмотря на то, что это может быть организовано с помощью последовательной записи вложенных операторов `if`, во многих случаях более эффективным оказывается применение оператора `switch`. Ниже приведена общая форма оператора.

`switch` (*выражение*)

```
{  
    case константа 1:  
        последовательность операторов блока 1  
        break;  
    case константа 2:  
        последовательность операторов блока 2  
        break;  
    . . .  
    default  
        последовательность операторов блока n  
        break;  
}
```

Этот оператор работает следующим образом. Значение *выражения* последовательно сравнивается с *константами*. Как только

будет обнаружено совпадение, выполняется оператор или *последовательность операторов*, связанных с этим совпадением, до оператора break. Оператор break передаёт управление оператору, следующему за конструкцией switch. Если совпадений нет, то выполняется последовательность операторов, следующая после оператора default. Эта ветвь не является обязательной.

При использовании конструкции switch действуют следующие правила:

- *выражение* в конструкции switch должно быть целочисленного типа (char, byte, short или int) перечислимого типа или же типа строкового;
- нельзя использовать числа с плавающей точкой;
- *константы* оператора case должны иметь тот же тип, что и *выражение* в конструкции switch;
- в одном операторе switch не допускается наличие двух одинаковых по значению *констант*;
- допускается использовать одну и ту же последовательность операторов, в этом случае оператор *break* не записывается.

### Пример 13.

```
int n;  
m1: Console.WriteLine("Введите целое число");  
int a = int.Parse(Console.ReadLine());  
switch (a)  
{  
    case 1:  
        n = 10;  
        break;
```

```

    case 2:
    case 3:
        n = 20;
        break;
    default:
        n = 0;
        break;
}
Console.WriteLine("a = " + a + "    n = " + n);
if (a != 0) goto m1;
Console.Read();

```

В данном примере в программу вводится и присваивается переменной **a**, любое целое число. С помощью конструкции **switch** происходит анализ. Если переменная **a** имеет значение равное 1, переменной **n** присваивается значение 10 и далее следует вывод этих переменных. Если **a** имеет значение равное 2 или 3, то переменной **n** присваивается значение 20 и далее вывод этих переменных. Во всех остальных случаях переменной **n** присваивается значение 0. Программа продолжает работать до тех пор, пока переменной **a**, не будет задано значение 0.

Один оператор **switch** может быть частью последовательности другого внешнего оператора **switch**. Такой оператор называется вложенным. Константы внешнего и внутреннего операторов **switch** могут содержать общие значения, не вызывая каких либо конфликтов.

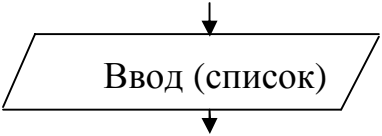
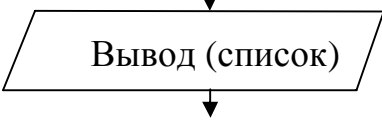
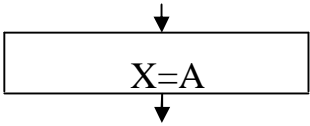
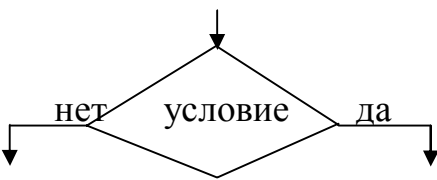
В операторе **switch** отсутствует возможность задания диапазона выбора, что является его недостатком. Например, в языке

программирования Visual Basic в аналогичном операторе задание диапазона выбора допускается.

### Разветвляющиеся программы

Разветвляющиеся программы это – такие программы, в которых на определённых этапах происходит анализ значений тех или иных параметров и в зависимости от этого выбирается один из возможных вариантов дальнейшего хода программы. Практически все более или менее сложные программы являются разветвляющимися. Для их написания используются рассмотренные конструкции управляющих операторов принятия решения.

Таблица 3.1

№	Название блока	Графическое изображение блока	Операторы и функции эквивалентные блоку
1	Блок ввода		Операторы ввода, функция InputBox и другие
2	Блок вывода		Операторы вывода, функция MsgBox и другие
3	Блок присваивания		Оператор присваивания
4	Блоки сравнения		Условный оператор i f



При написании разветвляющих программ предварительно составляется блок-схема алгоритма решения задачи. Блок-схема это – графическое изображение алгоритма или последовательности решения задачи программирования.

Для составления блок-схем используются стандартизованные графические изображения (блоки) определённых операторов алгоритмического языка. Некоторые из них представлены в таблице 3.1.

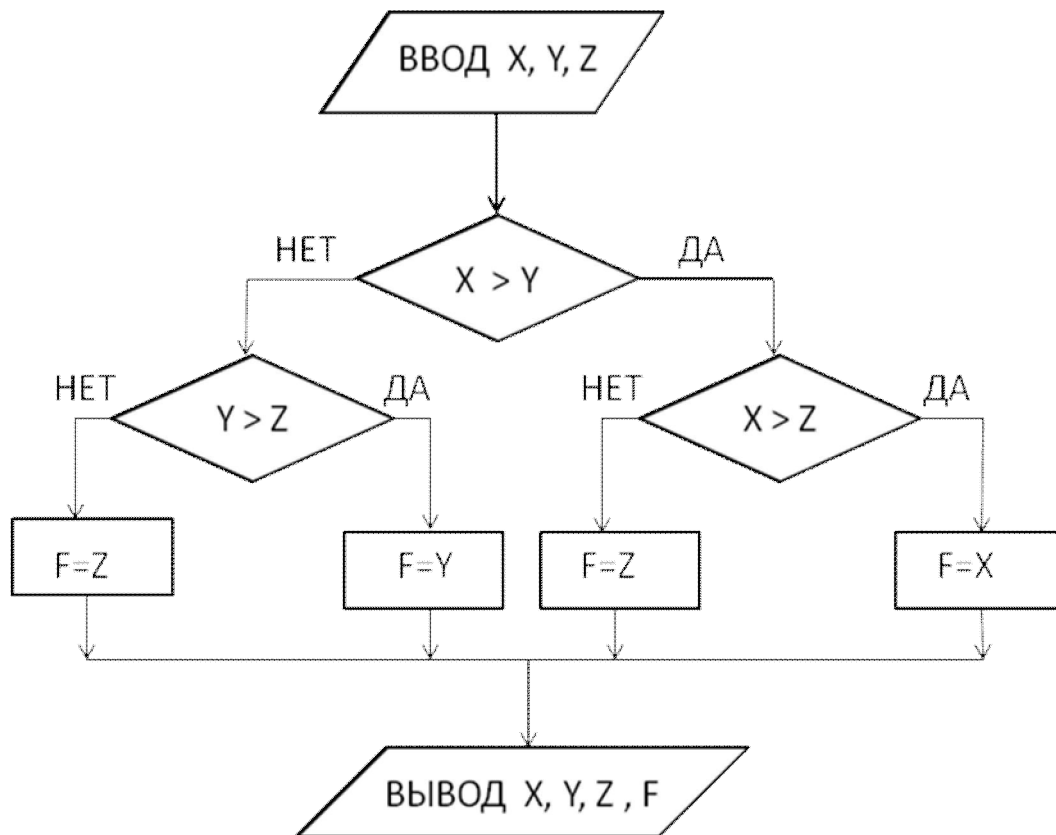
Далее рассмотрены примеры написания разветвляющихся программ, аналогичные тем, которые должен выполнить студент в данной лабораторной работе.

#### **Пример 14.**

Составить блок-схему и написать программу для определения наибольшей из трёх заданных величин  $X$ ,  $Y$  и  $Z$ . Полученное значение присвоить переменной  $F$ , т. е. вычисляет  $F = \max(X, Y, Z)$ . Замечание: Данный пример является тренировочным, на практике подобные задачи решаются с помощью соответствующих встроенных функций.

Пояснения к блок-схеме. После ввода численных значений для переменных  $X$ ,  $Y$  и  $Z$  производится их последовательное сравнение друг с другом на предмет выявления наибольшего из них. Первоначально сравниваются значения переменных  $X$  и  $Y$ . Если условие  $X > Y$  выполняется (истинно), то далее переменная с наибольшим значением, а именно  $X$  сравнивается с  $Z$ . Если поставленное в блоке сравнения условие  $X > Z$  верно, то переменной  $F$  будет присвоено значение переменной  $X$  в противном случае – значение переменной  $Z$ . Аналогично поступаем в случае если условие  $X > Y$ , не выполняется (ложно).

## Блок-схема



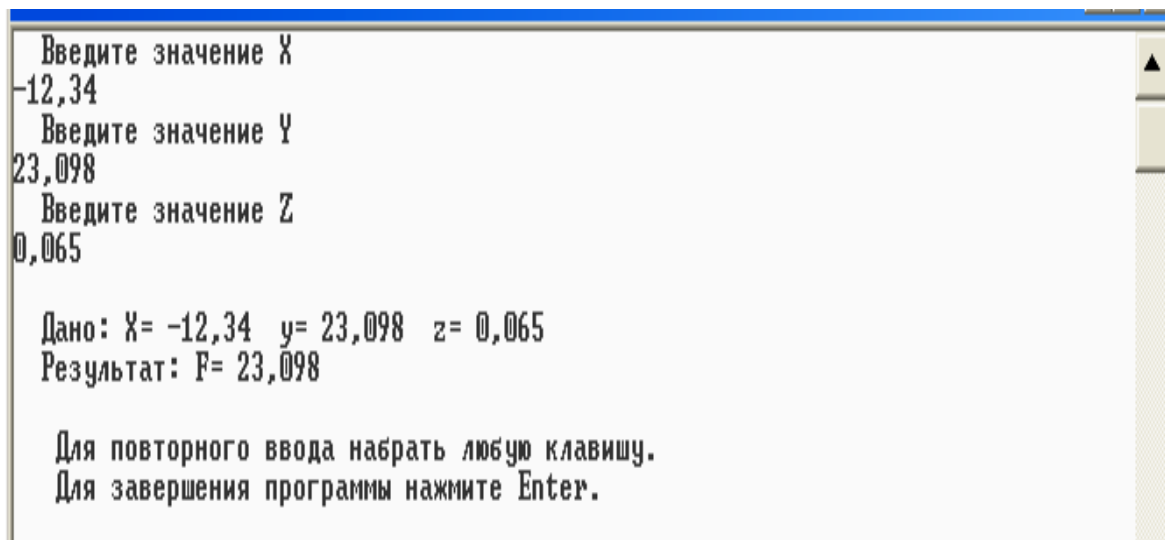
После составления блок-схемы по ней пишется программа, при этом каждый блок описывается соответствующим оператором алгоритмического языка.

```
float f;  
m1: Console.WriteLine(" Введите значение X");  
float x = float.Parse(Console.ReadLine());  
Console.WriteLine(" Введите значение Y");  
float y = float.Parse(Console.ReadLine());  
Console.WriteLine(" Введите значение Z");  
float z = float.Parse(Console.ReadLine());  
if (x > y)
```

```

{
    if (x > z) f=x;
    else f=z;
}
else
{ if(y>z) f=y; else f=z; }
Console.WriteLine('\n' + " Дано: X= " +
x + " y= " + y + " z= " + z +
'\n' + " Результат: F= " + f);
Console.WriteLine('\n' + "Для повторного
ввода" + "набрать любую клавишу." +
'\n' + "Для завершения программы нажмите
Enter.");
string p = Console.ReadLine();
if (p != "") goto m1;

```



```

Введите значение X
-12,34
Введите значение Y
23,098
Введите значение Z
0,065

Дано: X= -12,34 y= 23,098 z= 0,065
Результат: F= 23,098

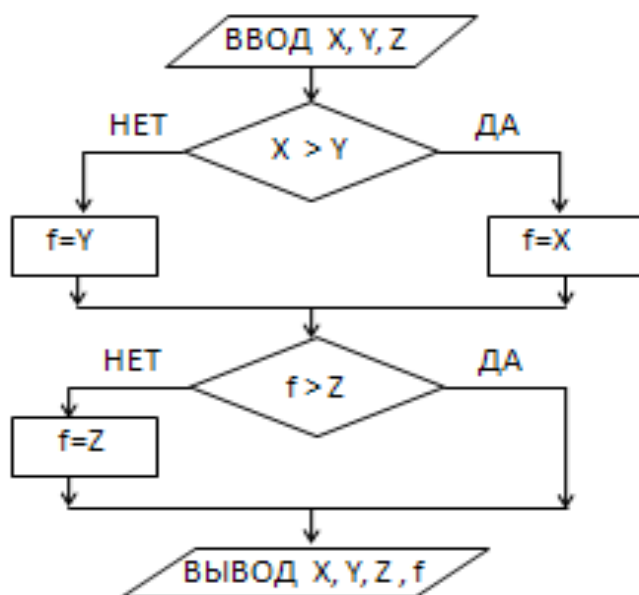
Для повторного ввода набрать любую клавишу.
Для завершения программы нажмите Enter.

```

Рассмотренный алгоритм решения задачи примера 14 не является единственным. Ниже представлена блок-схема другого вари-

анта алгоритма и основной фрагмент программы, с использованием оператора проверки

### Блок-схема



```

. . .
f = (x > y) ? x : y;
f = (f > z) ? f : z;
. . .

```

### Основной фрагмент программы

### Пример 15.

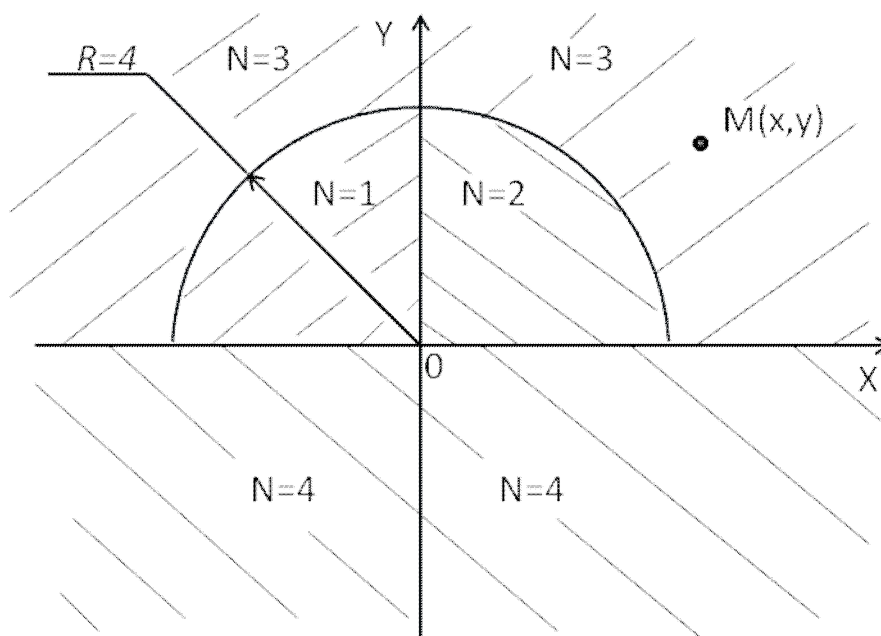
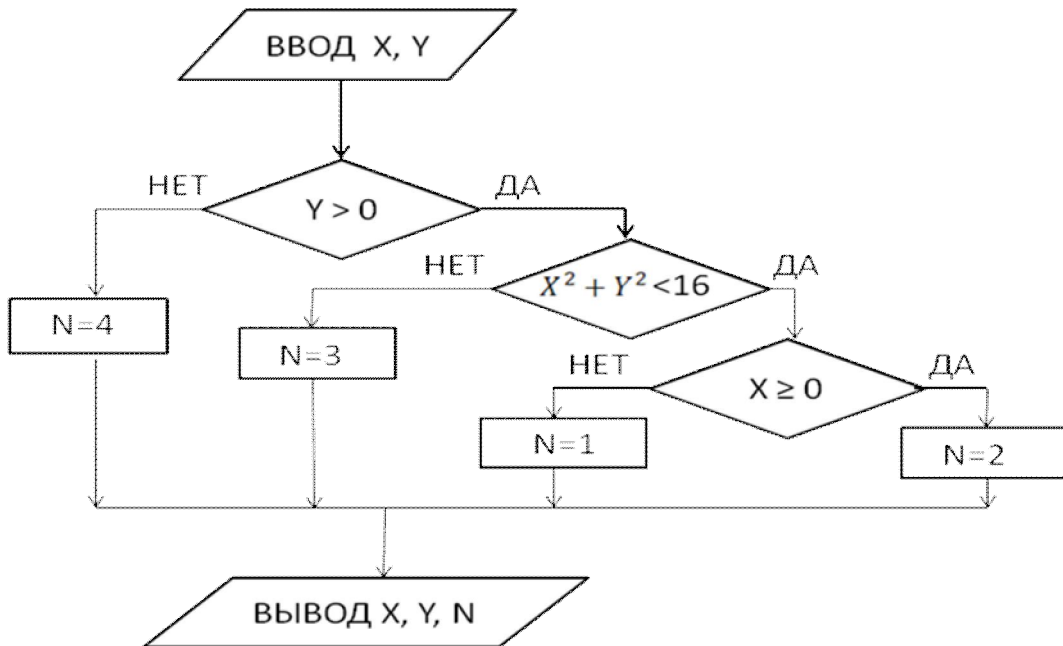


Рис. 3.1

**Задание.** Составить блок-схему и написать программу, которая определяет номер  $N$  области, в которой находится точка  $M(x, y)$  с заданными координатами (см. рисунок 3.1). Границы области относить к области с наибольшим номером.

### Блок-схема



Пояснения к блок-схеме. В первом блоке производится ввод численных значений для переменных  $X$  и  $Y$ , которые являются координатами точки  $M$ . Далее целесообразно сравнить переменную  $Y$  (координата по оси  $Y$ ) с нулём. В блок-схеме это первый блок сравнения, если его условие  $Y > 0$  не выполняется (ложно), то координата по оси  $Y$  точки  $M$  отрицательна или равна нулю, а это значит, что она расположена ниже оси  $X$  или на ней, т. е. в области с номером  $N = 4$ . Если условие  $Y > 0$  первого блока сравнения выполняется (истинно), то точка  $M$  расположена выше оси  $X$ , а это значит, что она может находиться в одной из областей с номером  $N = 1$ ,  $N = 2$  или  $N = 3$ . Далее для определения номера области

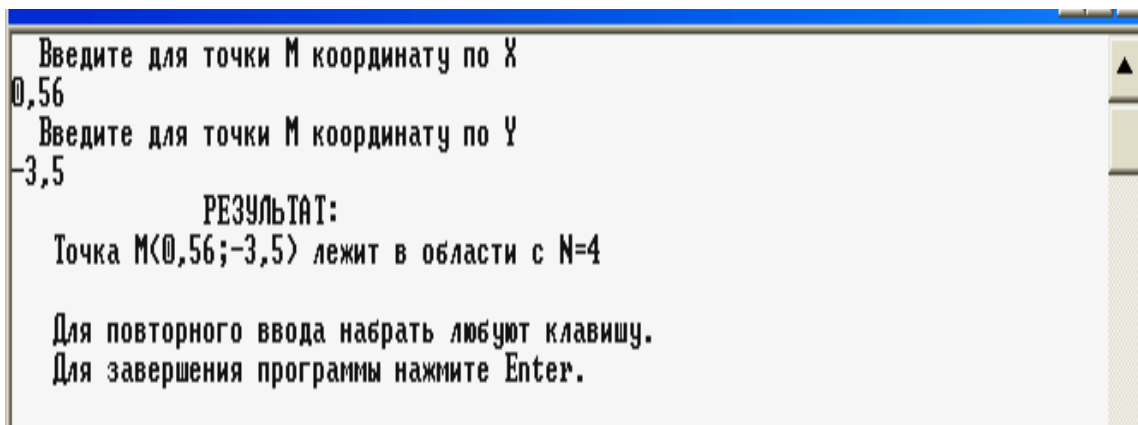
целесообразно задать во втором блоке сравнения условие  $X^2 + Y^2 < 16$ , которое следует из уравнения окружности  $X^2 + Y^2 = R^2$ , где радиус окружности. Если заданное условие выполняется, то точка расположена внутри окружности, а так как  $Y > 0$ , то внутри полуокружности. Согласно условию задачи внутри полуокружности точка может находиться либо в области с номером  $N = 1$ , либо в области с  $N = 2$ . Если условие  $X \geq 0$  третьего блока сравнения выполняется (истинно), то точка расположена в области с  $N = 2$ , в противном случае с  $N = 1$ . После чего идёт печать результата. Если условие  $X^2 + Y^2 < 16$  второго блока сравнения не выполняется (ложно), то точка  $M$  находится вне полуокружности и над осью  $X$  так как  $Y > 0$  т. е. в области  $N = 3$ . Далее представлена программа, составленная по рассмотренной блок-схеме.

```
int N;
m2: Console.WriteLine(" Введите для точки M"+
    " координату по X ");
float x = float.Parse((Console.ReadLine()));
Console.WriteLine(" Введите для точки M" +
    " координату по Y ");
float y = float.Parse((Console.ReadLine()));
if (y > 0)
{
    if (x * x + y * y < 16)
    {
        if (x >= 0) N = 2;
        else N = 1;
    }
}
```

```

        else
            { N = 3; }
    }
else
    { N = 4; }
Console.WriteLine('\t' + "        РЕЗУЛЬТАТ:");
Console.WriteLine("    Точка М(" + x + "; " + y + " )"
+ " лежит в области с N=" + N);
Console.WriteLine('\n' + "    Для повторного вво
да" + " нажать любую клавишу. " + '\n' +
"    Для завершения программы нажмите Enter.");
string p = Console.ReadLine();
if (p != "") goto m2;

```



Результаты расчёта по программе примера 15

## 2. Практическая часть

### Задания к лабораторной работе

Составить блок-схему и написать программу для выполнения следующих заданий. При этом руководствоваться выше приведёнными примерами выполнения заданий (см. примеры 14 и 15)

**Задание 1.** Вычислить для своего варианта значение функции  $F$ . При получении в знаменателе нуля дать соответствующее сообщение.

### Варианты заданий

$$1) \quad F = \frac{\min(x, y) + 0,5}{(\max(x, y))^2 - \sin z}$$

$$2) \quad F = \frac{\min(x, y, z) + x}{(\max(x, z))^2 + y}$$

$$3) \quad F = \frac{\max(x, y) + y}{(\min(x, y, z))^2 + yx}$$

$$4) \quad F = \frac{\min(z, \max(x, y))}{x^2 + z}$$

$$5) \quad F = \frac{\max(x^2, y^2, xz) + x}{(\min(x, y))^2 - y}$$

$$6) \quad F = \frac{\min(x, y + z)}{\max(x^2, y) + z^3}$$

$$7) \quad F = \frac{\max(x^2, y^2, x - y) + x}{(\min(x, y))^2 + y^4}$$

$$8) \quad F = \frac{\min(x, (x + y)^2)}{x^2 + \max(y^3, x)}$$

$$9) \quad F = \frac{\max(x + z, \min(x, y))}{x^2}$$

$$10) \quad F = \frac{\min(x, \max(x + y, z))^2}{x^2 + z^2}$$

$$11) \quad F = \frac{\min(x, y + z)}{\max(x, y) + \sin z}$$

$$12) \quad F = \frac{\min(x, y - x)}{\max(yz, x^2) + \cos 2z^3}$$

$$13) \quad F = \frac{\max(x^2, z^2) + \cos 2x^2}{(\min(x, y))^2 - y}$$

$$14) \quad F = \frac{\min(x^2, y + z)}{x^2 + \max(z^3, xy)}$$

$$15) \quad F = \frac{\max(x + y, \max(x, zy))}{xe^2}$$

$$16) \quad F = \frac{\max(x^3, y^2, xy) + x}{(\min(x, yz))^2 - y}$$

$$17) \quad F = \frac{x(\max(x + z, zy))}{\min(x, y) + x^2} 1$$

$$18) \quad F = \frac{\min(x, \max(x + z, y))^2}{x^3 + z^2}$$

$$19) \quad F = \frac{x^3 + \max(z^2, y)}{(\max(x, z))^2 - y}$$

$$20) \quad F = \frac{\max(x, y + z) + e^{xz}}{\min(x^2, y) + z^3}$$



$$21) F = \frac{\min(x^2, z^4, xy) + x}{(\max(x, y))^2 - y}$$

$$22) F = \frac{\min(x, y+z) + e^x}{\max(x^2, y) + z^3}$$

$$23) F = \frac{\max(x^2, y^2) + \cos 4z^2}{\min(x, y) + x^2}$$

$$24) F = \frac{(\min(x, y))^2 - y}{x^2 + \max(z^3, x)}$$

$$25) F = \frac{\min(x, y+2x)}{\max(y, z) + \sqrt[3]{x}}$$

$$26) F = \frac{\min(x, y-z)}{\max(yz, x^2) + \cos 2x^2}$$

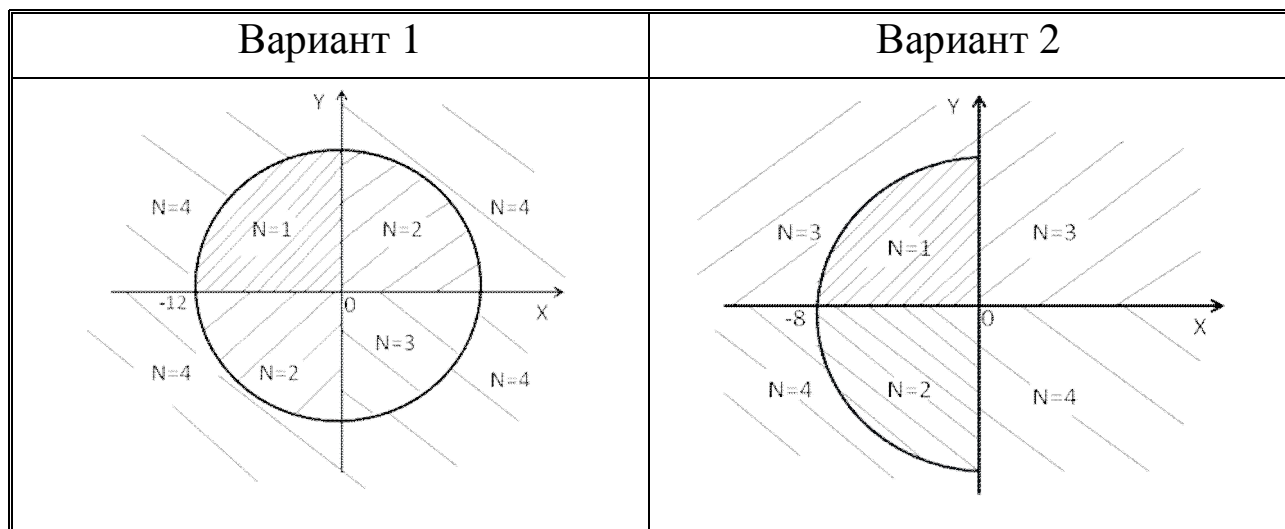
$$27) F = \frac{\max(x^3, z^2) + \cos 4y^2}{\min(y, yz) + \sqrt{x}}$$

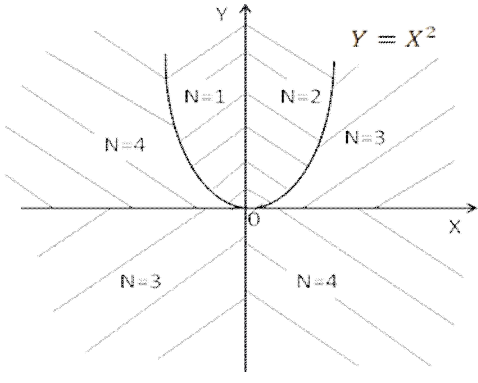
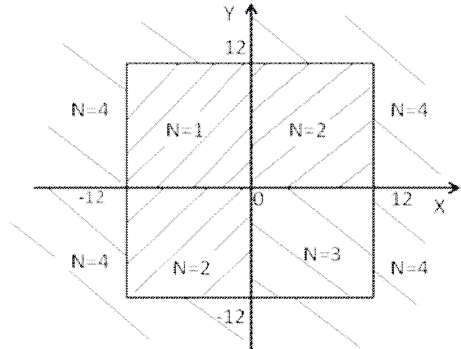
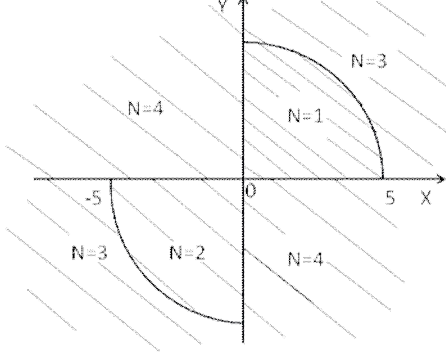
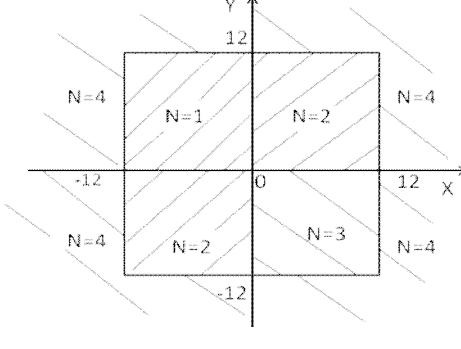
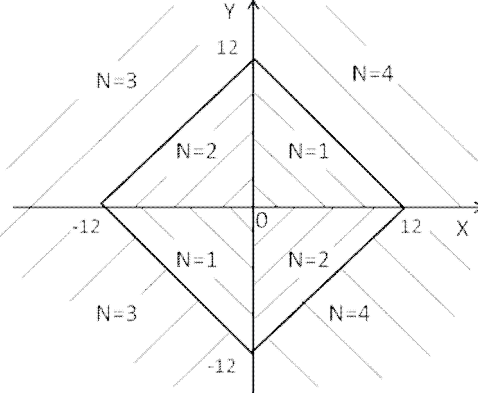
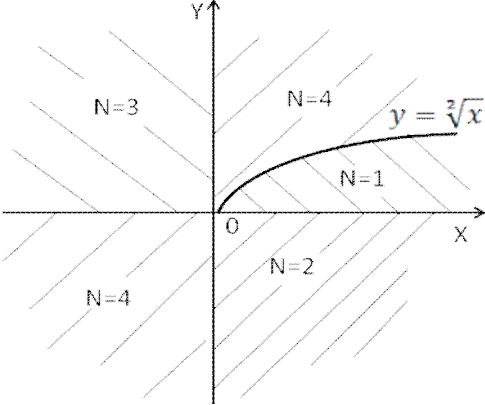
$$28) F = \frac{(\min(x, y))^4 + 2e^x}{x^3 + \max(z^2, x)}$$

$$29) F = \frac{\max(xz, \min(y, z))}{x^2 + \sin zy}$$

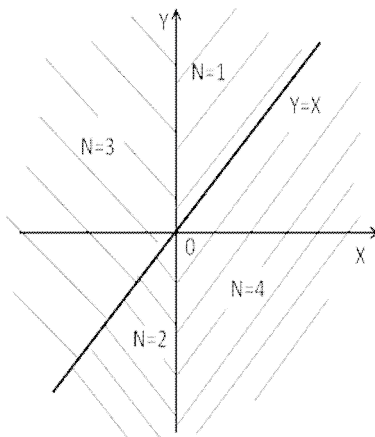
$$30) F = \frac{\max(x, \max(y, z))^4}{\sin 2y + xe^2}$$

**Задание 2.** Определить для своего варианта номер N области, в которой находится точка M(x,y) с заданными координатами. Границы области относить к области с наибольшим номером.

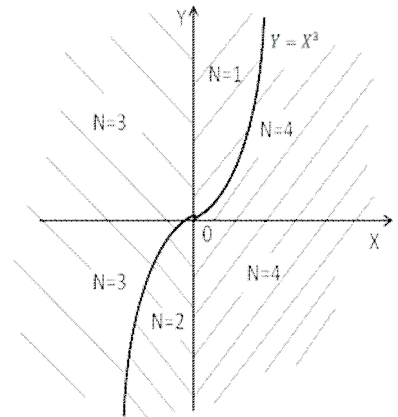


Вариант 3	Вариант 4
	
Вариант 5	Вариант 6
	
Вариант 7	Вариант 8
	

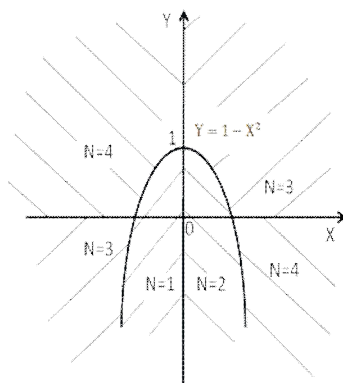
Вариант 9



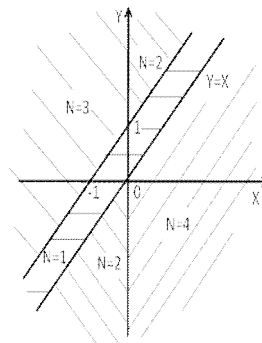
Вариант 10



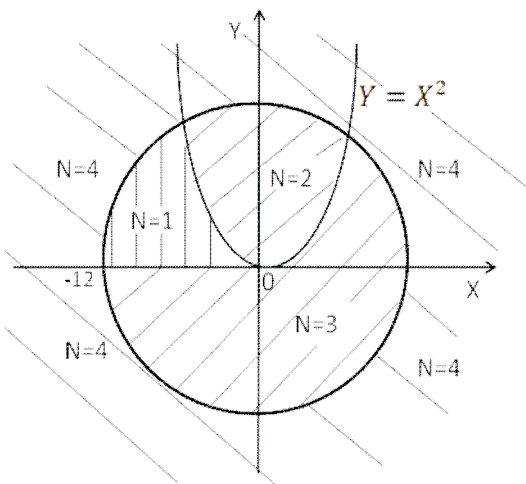
Вариант 11



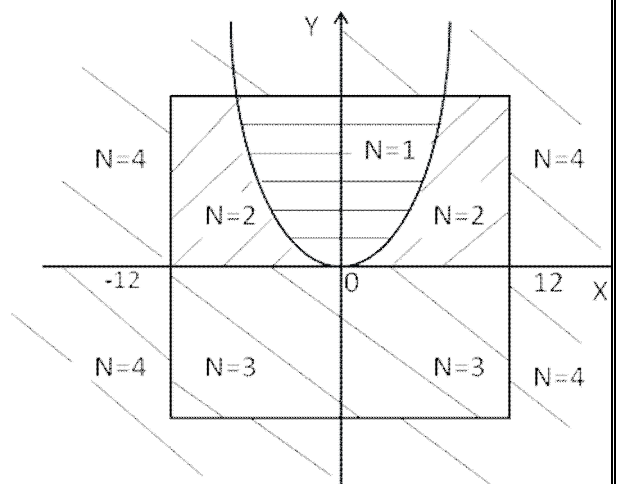
Вариант 12



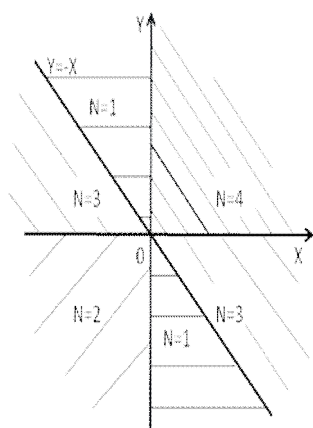
Вариант 13



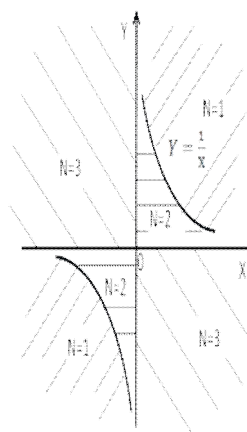
Вариант 14



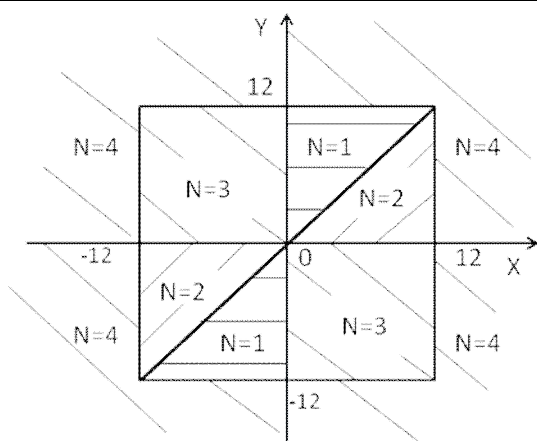
Вариант 15



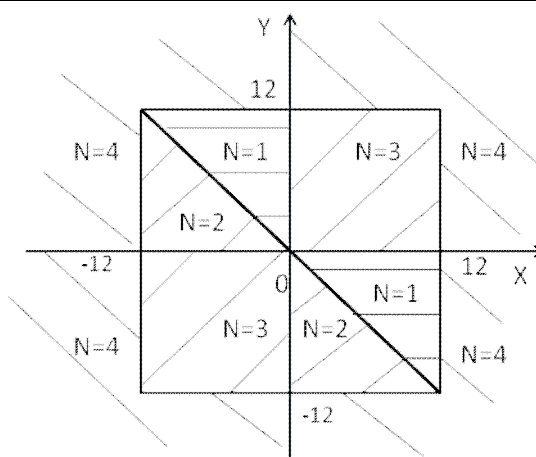
Вариант 16



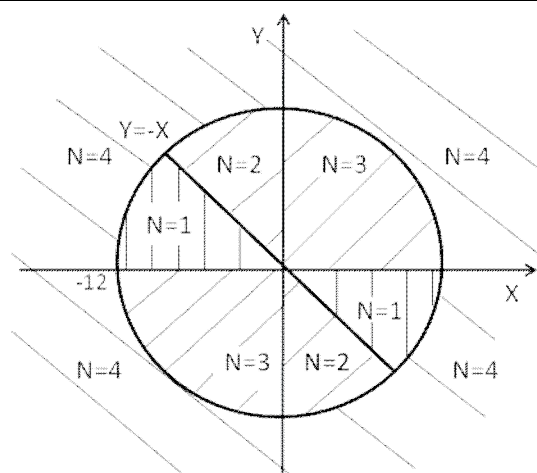
Вариант 17



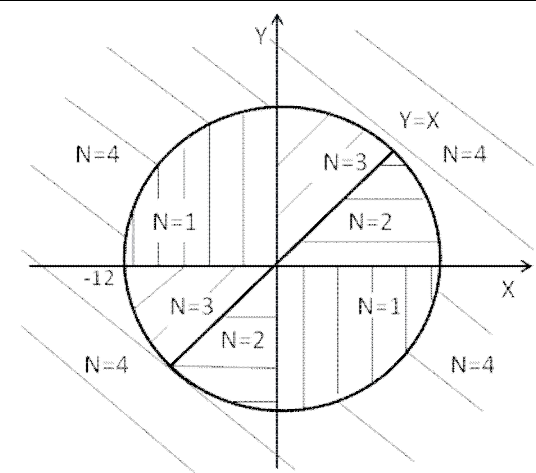
Вариант 18



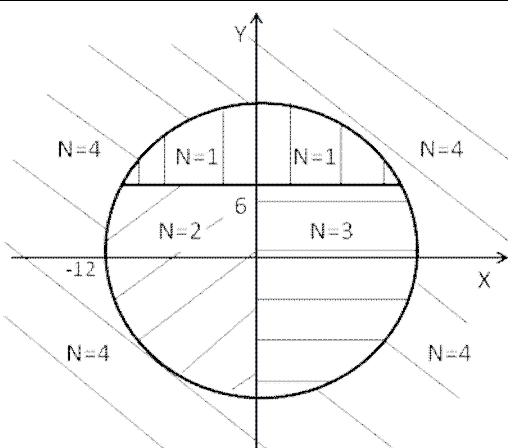
Вариант 19



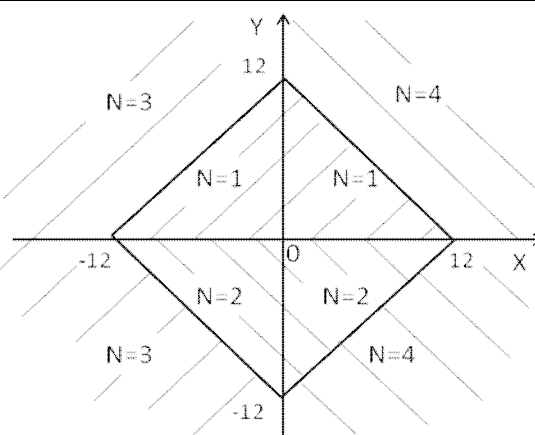
Вариант 20



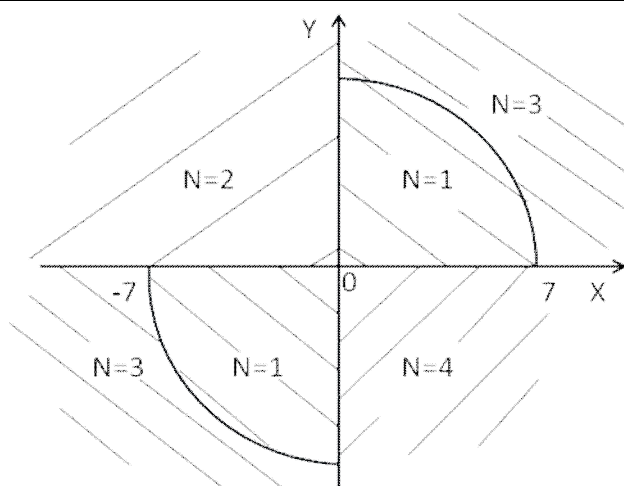
Вариант 21



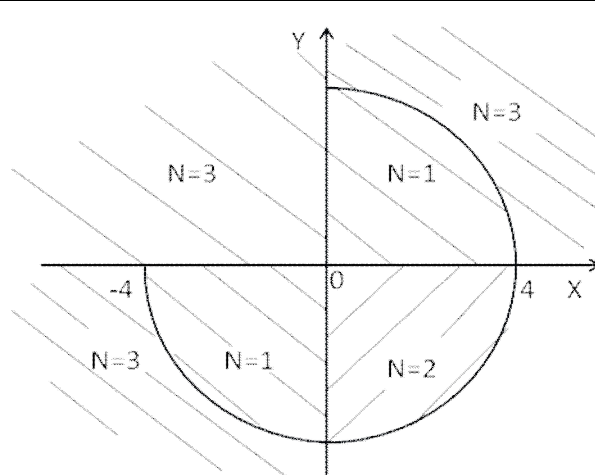
Вариант 22



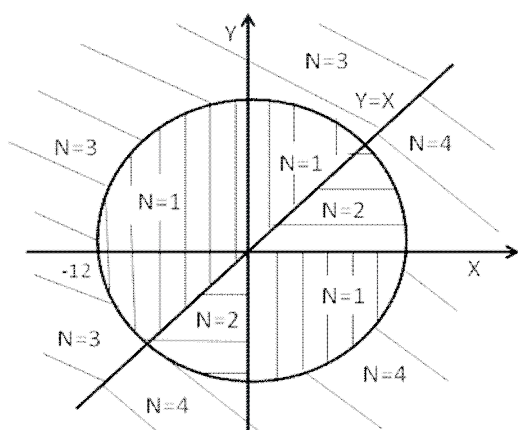
Вариант 23



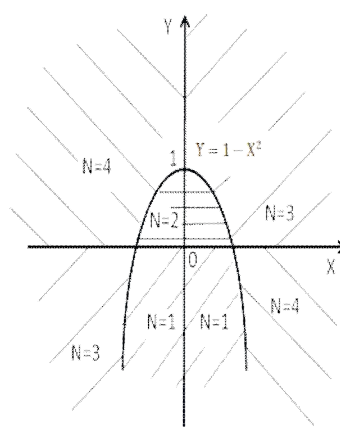
Вариант 24



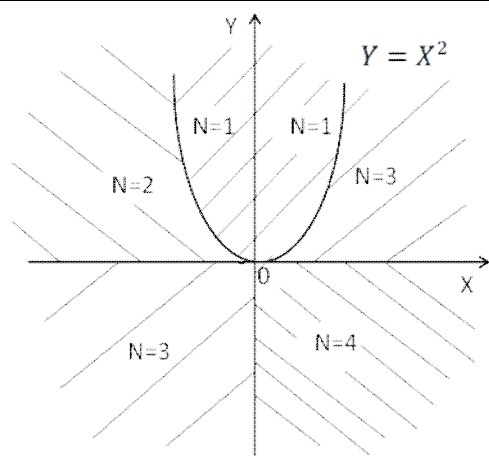
Вариант 25



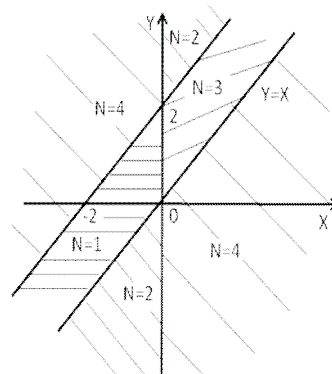
Вариант 26



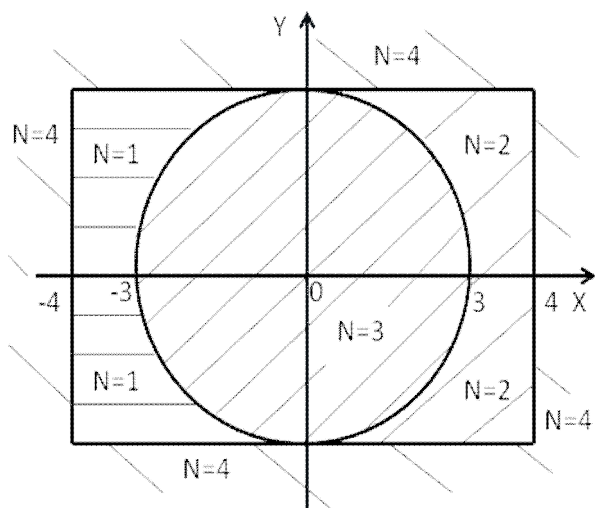
Вариант 27



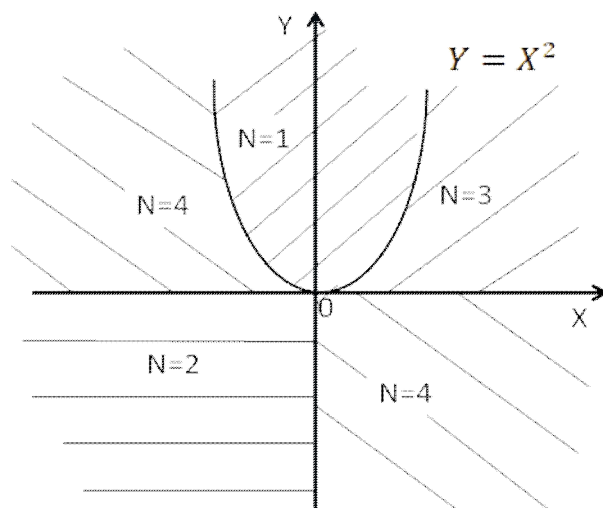
Вариант 28



Вариант 29



Вариант 30



## ЛАБОРАТОРНАЯ РАБОТА № 4

### Управляющие операторы для организации циклов.

#### Программы с циклами

#### 1. Краткие теоретические сведения

Циклом называется процесс исполнения группы операторов программы заданное количество раз, либо до тех пор, пока выполняется или не выполняется некоторое поставленное условие. Процесс исполнения группы операторов цикла один раз называется итерацией цикла. Группа операторов, расположенная между началом и концом цикла называется *телом цикла*. В С# есть три основных вида циклов: цикл `for` с параметром (счётчиком), цикл `while` с предусловием и цикл `dowhile` с постусловием.

#### Оператор цикла `for`

Оператор цикла `for` (для) служит для организации циклов с параметром (счётчиком). Это наиболее распространённый оператор цикла. Он проще не только для чтения и понимания, но и проверки корректности цикл. Оператор цикла `for` имеет следующую структуру.

```
for (выражение 1; условие; выражение 2)  
{  
    // тело цикла  
}
```

Открывающиеся и закрывающаяся фигурная скобки являются соответственно началом и концом тела цикла. В круглых скобках указаны следующие параметры цикла:

- *выражение 1*, в нём указывается имя переменной управления циклом и присваивается ей начальное значение;
- *условие*, представляет собой булево выражение, проверяющее значение переменной управления циклом, если результат проверки истинен, то цикл продолжается, если ложен, то цикл завершается;
- *выражение 2*, как правило, это арифметическое выражение, счётчик, определяет на каждой итерации цикла порядок изменения переменной управления циклом, на определённую величину – шаг цикла.

При работе цикла сначала выполняется *выражение 1*. В результате переменная управления циклом принимает своё первоначальное значение. Затем вычисляется *условие* и, если оно истинно, выполняются операторы тела цикла, заключённые в фигурные скобки. По достижению конца тела цикла (закрывающаяся фигурная скобка) управление переходит к вычислению *выражение 2*. В результате переменная цикла принимает новое значение, после чего вновь вычисляется *условие* и в случае его истинности цикл повторяется, в противном случае управление передаётся первому оператору, следующему за закрывающейся фигурной скобкой.

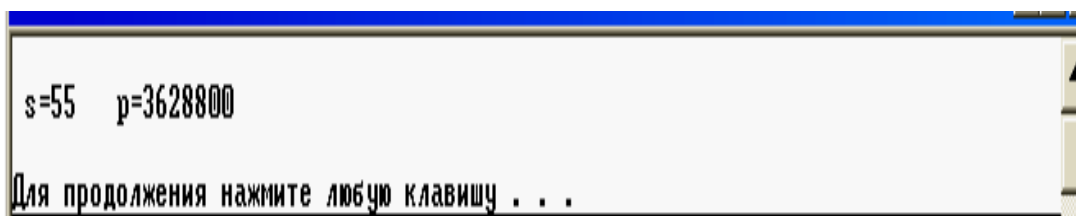
### **Пример 1.**

```
int s, p, n;
s = 0;
p = 1;
for (n = 1; n <= 10; n = n + 1)
{
    s = s + n;
    p = p * n;
```



```
}
Console.WriteLine(" s=" + s + " p=" + p);
```

В данной программе определяется сумма  $S$  и произведение  $p$  чисел от одного до десяти включительно. Имя переменной управления циклом  $n$ , её заданное начальное значение 0. Условие продолжения цикла  $n \leq 10$ . Порядок изменения переменной управления циклом, счётчик –  $n = n + 1$ . Таким образом,  $n$  изменяется от 1 до 10 с шагом 1. При этом на каждой итерации сумма  $S$  увеличивается на  $n$ , а произведение  $p$  в  $n$  раз. Ниже представлены результаты работы программы.

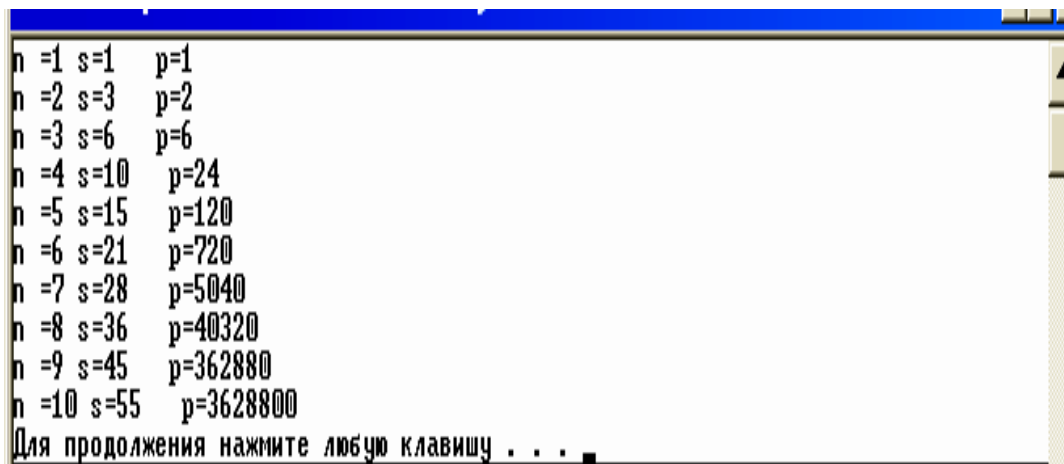


Если в рассмотренной программе вывод организовать внутри цикла, то можно проанализировать увеличение сумм  $S$  и произведения  $p$  на каждой итерации цикла. Кроме того можно записать программу более компактно см. пример 2.

### **Пример 2.**

```
int s, p, n;
s = 0;
p = 1;
for (n = 1; n <= 10; n++) // или n += 1
{
    s = s + n;
    p = p * n;
    Console.WriteLine("n =" + n + "s=" + s + "p="
```

```
+p);  
}
```



n =1	s=1	p=1
n =2	s=3	p=2
n =3	s=6	p=6
n =4	s=10	p=24
n =5	s=15	p=120
n =6	s=21	p=720
n =7	s=28	p=5040
n =8	s=36	p=40320
n =9	s=45	p=362880
n =10	s=55	p=3628800

Для продолжения нажмите любую клавишу . . .

Результаты работы данной программы

Поскольку добавление 1 к переменной, в частности к  $n = n + 1$ , является распространённой операцией в языке C# предусмотрена сокращённая запись этой операции, а именно  $n ++$ , как в примере 2, или  $n += 1$ . Этот оператор называется оператором инкремента, он часто используется как в цикле `for` так и в других операторах цикла.

В теле цикла `for` нельзя изменять его параметры. Если в цикле отсутствует параметр *условие*, то это соответствует значению `true`. Цикл, записанный в виде `for(;;)` – является бесконечным циклом.

### Оператор цикла `while`

Оператор цикла `while` (пока), как и оператор цикла `for`, является циклом с предусловием. Это означает, что *условие* проверяется до начала цикла, и если оно имеет значение `false`, то цикл ни

разу не выполняется. Отличие оператор цикла `while` от оператора цикла `for` состоит в том, что в нём заранее не определено количество итераций в цикле. Форма записи оператора.

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

Операторы в теле цикла исполняются до тех пор, пока *условие* цикла выполняется, т.е. имеет значение `true`. Если вместо *условия* указано служебное слово `true`, т.е. `while (true)`, то цикл будет бесконечным.

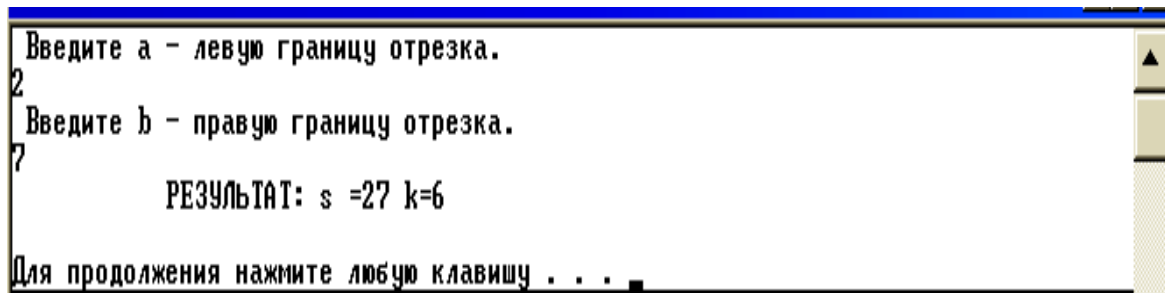
**Пример 3.** Составить программу, которая определяет сумму  $S$  и количество  $K$  целых чисел лежащих на заданном отрезке  $[a; b]$ . Границы отрезка целые числа.

```
int a, b, s, k;
Console.WriteLine(" Введите a" +
"- левую границу отрезка.");
a = int.Parse(Console.ReadLine());
Console.WriteLine(" Введите b" +
"- правую границу отрезка.");
b = int.Parse(Console.ReadLine());
s = 0;
k = 0;
while ( a <= b)
{
    S += a;          // означает S = S + a
    k++;             // означает k = k+1
}
```

```

    a++;          // означает a = a + 1
}
Console.WriteLine('\t' + "    РЕЗУЛЬТАТ: " +
    " s =" + s + " k=" + k + '\n');

```



Переменная *a* управления циклом имеет начальное значение равное левой границе заданного отрезка. Перед каждым шагом она сравнивается с переменной *b*, которая имеет значение правой границы заданного отрезка. Если результат сравнения *true*, то цикл продолжается. Сумма чисел отрезка, переменная *S*, увеличивается на величину значения переменной *a*, которая является очередным числом заданного отрезка. Количество чисел отрезка, переменная *k*, и переменная *a* увеличиваются на единицу. Если результат сравнения *false*, то цикл заканчивается, и выдаются, например следующие результаты работы программы.

### Оператор цикла `do-while`

Оператор цикла `do-while` является версией цикла `while` с постусловием. Это означает, что *условие* цикла проверяется после исполнения операторов тела цикла. Следовательно, в таком цикле

одна, первая итерация, всегда будет выполняться. Иногда это удобно использовать. Форма записи оператора.

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

После исполнения первой итерации цикла цикл продолжает работать до тех пор, пока *условие* цикла имеет значение true, в противном случае цикл заканчивается. В случае использования данного оператора в программе примера 3 она будет выглядеть следующим образом, см. пример 4.

#### **Пример 4.**

```
int a, b, s, k;
Console.WriteLine(" Введите a"+
" - левую границу отрезка");
a =int.Parse((Console.ReadLine()));
Console.WriteLine(" Введите b"+
" - правую границу отрезка");
b = int.Parse((Console.ReadLine()));
s = 0;
k = 0;
do
{s += a; k++; a++; }
while (a <= b);
Console.WriteLine(' \t' + "    РЕЗУЛЬТАТ: " +
" s =" + s + " k=" + k + '\n');
```

## **Операторы break и continue**

Для управления циклом имеются специальные операторы – break и continue. Оператор break вызывает прекращение выполнения цикла и передачу управления первому оператору, следующему непосредственно за циклом. Оператор continue передаёт управление в начало цикла, к проверке условия. Обычно эти операторы используются совместно с оператором if.

## **Вложенные циклы**

Цикл можно размещать внутри другого цикла. Размещение одной конструкции цикла в другой называется вложением циклов, а сам цикл вложенным.

При организации циклов, в том числе и вложенных, необходимо соблюдать следующие правила:

- переменная, объявленная в теле цикла, определена только внутри этого цикла;
- вход в цикл осуществляется через его заголовок;
- вход во внутренний цикл осуществляется после входа во внешний;
- выход из цикла, в том числе и вложенного возможен в любом месте тела цикла;
- параметры циклов со счётчиком не должны изменяться в теле цикла;
- вложенные циклы не должны пересекаться, т.е. начало и конец внутреннего цикла должны находиться во внешнем цикле;
- допускается делать вложение разных конструкций циклов.

### Пример 5. (пример выполнения задания 1)

Написать и отладить программу вычисления значений функции  $y = \frac{ax}{\sqrt{1+ax^2}}$ , для каждого из заданных значений параметра  $a$  (0, 5; 1, 0; 1, 5; 2, 0) и при всех заданных значениях аргумента  $x$  (от 1 до 7 с шагом 0, 25).

```
StreamWriter p = new StreamWriter("rez.txt");
float a, x, y;
p.WriteLine("          РЕЗУЛЬТАТЫ РАСЧЁТА");
for (a = 1; a <= 2; a += 0.5f) // Заголовок внешнего
                               // цикла по a
{
    // Начало внешнего цикла по a
    p.WriteLine(" a=" + a);
    for (x = 1; x <= 7; x += 0.25f) //Заголовок
                                    // внутреннего
                                    // цикла по x
    {
        // Начало внутреннего цикла по x
        y = (float)(a * Math.Cos(x) /
        Math.Sqrt(1 + a * x * x));
        p.WriteLine("      x= " + x +
                    '\t' + "      y= " + y);
        /* Конец внутреннего цикла по x */
    }
    /* Конец внешнего цикла по a */
}
p.Close();
```

## Результаты расчёта

(файл rez.txt)

РЕЗУЛЬТАТЫ РАСЧЁТА	
a=1	
x= 1	y= 0,3820514
x= 1,25	y= 0,1969803
x= 1,5	y= 0,03923794
...	
x= 6,75	y= 0,1308689
x= 7	y= 0,1066179
a=1,5	
x= 1	y= 0,5125757
x= 1,25	y= 0,2586599
x= 1,5	y= 0,05072828
...	
x= 6,75	y= 0,1608578
x= 7	y= 0,1310171
a=2	
x= 1	y= 0,6238874
x= 1,25	y= 0,310508
x= 1,5	y= 0,06032489
...	

## Суммирование рядов

Выше было рассмотрено использование циклических конструкций для вычисления, накапливания сумм (примеры 1 – 4). Накапли-



вание сумм используется при вычислении, суммировании рядов, их членов. С помощью рядов могут вычисляться различные функции. Рассмотрим такую задачу на примере.

**Пример 6.** (пример выполнения задания 2)

Написать и отладить процедуру для приближённого вычисления функции  $y = \sin(x)$  с помощью ряда

$$s(x) = x - \frac{x^3}{3!} + \frac{x^5}{5!} - \frac{x^7}{7!} + \dots$$

Суммирование членов ряда проводить включительно до члена ряда, значение которого по абсолютной величине меньше чем  $10^{-6}$ .

Вычислить сумму членов заданного ряда для угла в 30 градусов и сравнить полученное значение со значением, вычисленным непосредственно, с помощью функции  $y = \sin(x)$ .

**Программа примера 6**

```
double g, pi, x, y, s, a, p, d, чл, зн;
int n, k;
m1: Console.WriteLine(" Введите значение угла в
градусах");
g = double.Parse((Console.ReadLine()));
pi = Math.PI; // Задание числа pi
x = g*pi/180; // Перевод градусов в радианы.
Console.WriteLine('\t' + "Промежуточные результаты"
+' \n' );

K = 0; // Номер итерации.
S = 0; p = 1;
n = 3; d = 1;
a = x; // Первый член ряда.
```

```

while (Math.Abs(a)>= 0.000001)
{
    S += a;           // Сумма ряда на текущей
                      // итерации.
    чл = x*x*x*d;     // Вычисление числителя ряда.
    p = -p*(n-1)*n;    // Вычисление факториала.
    зн = p;           // Знаменатель ряда.
    a = чл/зн;        // Член ряда на текущей итерации.
    d = x*x;
    n += 2;
    k++;              // Счётчик итераций.
    Console.WriteLine('\t' + " Итерация №"
+ k + '\n' + " a=" + a + " s=" + s + '\n');
}
y = Math.Sin(x);     // Непосредственное вычисление
                      // синуса.
Console.WriteLine('\t' + " РЕЗУЛЬТАТЫ: " + '\n'
+ " Заданное значение угла в град. =" + g + '\n'
+ " Вычисленная сумма ряда S =" + s + '\n' + "
Количество членов ряда - " + k + '\n'
+ " Функция SIN(X)=" + y + '\n');
Console.WriteLine('\n' + " Для повторного ввода"
+ " набрать любую клавишу. " + '\n'
+ " Для завершения программы нажмите Enter.");
string rep = Console.ReadLine();
if (rep != "") goto m1;

```

## Результаты расчёта по программе примера 6

```

Введите значение угла в градусах
30
Промежуточные результаты

Итерация №1
a=-0,023924596203935 s=0,523598775598299

Итерация №2
a=0,000327953194428671 s=0,499674179394364

Итерация №3
a=-7,80840939115883E-06 s=0,500002132588792

Итерация №4
a=1,08450130432761E-07 s=0,499994324179401

РЕЗУЛЬТАТЫ:
Заданное значение угла в градусах =30
Вычисленная сумма ряда S =0,499994324179401
Количество членов ряда - 4
Функция SIN(X)=0,5

Для повторного ввода набрать любую клавишу.
Для завершения программы нажмите Enter.

```

## 2. Практическая часть

**Задание 1.** Составить и отладить программу вычисления заданной в таблице 4.1 функции  $y(x)$  для каждого из заданных значений параметра  $a$  и при всех заданных значениях аргумента  $x$ .

Таблица 4.1

	Заданная функция $y(x)$	Значение аргумента $x_{нач}; x_{кон}; \Delta x$	Значение параметра $a$
1.	$y = (1/a) \cdot \exp(-(x/a)^2)$	-1,5; 1,5; 0,1	1; 1,1; 1,2; 1,3
2.	$y = x \cdot e^{-x/a}$	0; 4; 0,2	0,25; 0,5; 0,75; 1,0
3.	$y = 1/\sqrt{(1-x^2)^2 + 4a^2x^2}$	0; 2; 0,05	0,1; 0,2; 0,3; 0,4

4.	$y = x \cdot \operatorname{tga} - x^2 / \cos^2 a$	0; 0,5; 0,02	$15^0; 30^0; 45^0; 60^0$
5.	$y = a \cdot x^a \cdot e^{-x/a}$	0; 10; 0,25	1; 1,25; ... 2,0
6.	$y = e^{-xa} \cdot \sin x$	0; $\pi$ ; $\pi/36$	0; 0,5; ... 2,0
7.	$y = ((x+a)^{2/3} - (x-a)^{2/3})/a$	-4; 4; 0,2	1; 2; 3; 4
8.	$y = a^{-x} - a^{-a \cdot x}$	0; 2; 0,05	10; 8; 6; 4; 2
9.	$y = (1 - \exp(-(x/a)^2))/a$	-2; 2; 0,1	1,25; 1,5; 1,75; 2
10.	$y = a^3 / (a^2 + x^2)$	-2; 2; 0,05	0,5; 1,0; 1,5; 2,0
11.	$y = \frac{\sqrt[4]{(a+1)x + e^{-x^3}}}{\sqrt{2ax}}$	1; 7; 0,25	0,5; 0,75; 1,0; 1,25
12.	$y = \frac{\sqrt[3]{ax^2 + e^{-x^2}}}{\sqrt{(a+1)x}}$	1; 7; 0,25	0,5; 0,75; 1,0; 1,25
13.	$y = \frac{\sqrt[3]{x + ae^{-x^2}}}{\sqrt{ax}}$	1; 7; 0,25	0,5; 0,75; 1,0; 1,25
14.	$y = (\sqrt{x + ae^{-x^2}})/(ax^2)$	1; 7; 0,25	0,5; 0,75; 1,0; 1,25
15.	$y = \frac{\operatorname{arctg}(\frac{x^2}{2a})}{x^3 + a}$	0; 2,5; 0,1	0,5; 1,0; 1,5; 2,0
16.	$y = \frac{\operatorname{arctg}(\frac{x}{2a})}{x^2 + 2a}$	0; 3; 0,1	0,5; 0,75; 1; 1,25
17.	$y = \frac{ax}{a + \sqrt[3]{1 + x^2}}$	-3; 3; 0,2	0,5; 1,0; 1,5; 2,0
18.	$y = \operatorname{Cos}^2(2ax)/(3a)$	$0^0; 360^0; 6^0$	1; 1,25; 1,5; 2,0
19.	$y = \operatorname{Sin}^2(ax)/(a+2)$	$0^0; 360^0; 6^0$	1; 1,25; 1,5; 2,0
20.	$y = a \operatorname{Cos}(2x)/(a+4)$	$0^0; 360^0; 6^0$	1; 2; 3; 4
21.	$y = a^2 e^{-x} / (2 + a^2)$	-4; 4; 0,25	1; 2; 3; 4

22.	$y = \frac{\sqrt[3]{ax^2 + e^{-x}}}{ax^2}$	1; 7; 0,25	0,5; 1,0; 1,5; 2,0
23.	$y = \frac{\sqrt{x + a^2} e^{-x^2}}{a + x}$	1; 7; 0,25	0,5; 1,0; 1,5; 2,0
24.	$y = (x^2 + a)^{\frac{xa}{x-1}}$	0; 0,8; 0,05	0,5; 1,0; 1,5; 2,0
25.	$y = (x + 4a)^{\frac{ax}{x-1}}$	0; 0,8; 0,05	0,5; 1,0; 1,5; 2,0
26.	$y = \ln^2 \left  \frac{xa}{a+x} \right $	2; 12; 0,5	0,5; 1,0; 1,5; 2,0
27.	$y = \ln \left  \frac{ax}{1+ax} \right $	2; 12; 0,5	0,5; 1,0; 1,5; 2,0
28.	$y = -\ln \left  \frac{x+a}{1+x^2} \right $	1; 10; 0,5	1; 2; 3; 4
29.	$y = \frac{ax^2}{\sqrt{1 + ax^2}}$	-2; 2; 0,1	1; 2; 3; 4
30.	$y = -\ln \left  \frac{ax}{a+x} \right $	1; 11; 0,5	1; 2; 3; 4

**Задание 2.** Составить и отладить программу для приближённого вычисления заданной функции  $y(x)$  путём суммирования членов заданного её ряда  $s(x)$  см. таблицу 4.2. Суммирование членов ряда проводить до члена ряда, значение которого по абсолютной величине не будет превышать  $10^{-6}$ .

Вычислить сумму ряда  $s(x)$  и непосредственно функцию  $y(x)$  при указанных в таблице 4.2 контрольных значениях аргумента  $x$ . Сравнить и проанализировать полученные значения.

Таблица 4.2

№	Ряд	Контрольные значения аргумента x и функция y(x)
1	$S = 1 + \frac{x}{2} - \frac{1 \cdot x^2}{2 \cdot 4} + \frac{1 \cdot 3 \cdot x^3}{2 \cdot 4 \cdot 6} - \frac{1 \cdot 3 \cdot 5 \cdot x^4}{2 \cdot 4 \cdot 6 \cdot 8} + \dots$	-0.84; 1; 2; $y = \sqrt{x+1}$
2	$S = 1 + \frac{x}{1!} + \frac{x^2}{2!} + \frac{x^3}{3!} + \dots$	1; 10; -10; $y = e^x$
3	$S = \sum_{n=1}^{\infty} (-1)^{n-1} \frac{2^{2n-1}}{(2n)!} x^{2n}$	$\pi/6$ ; $13\pi/6$ ; $25\pi/6$ ; $y = \sin^2 x$
4	$S = 1 + \sum_{n=1}^{\infty} (-1)^{n-1} \frac{1 \cdot 3 \cdot 5 \cdot \dots \cdot (2n-3)}{n! \cdot 2^{3n}} (x-4)^n$	2; 1; 0; $y = 0.5 \cdot \sqrt{x}$
5	$S = \frac{x}{2} + \frac{x^2}{2^2} + \frac{x^3}{2^3} + \dots$	-1; 1; 1.9; $y = \frac{x}{2-x}$
6	$S = \frac{x}{1} + \frac{1 \cdot x^3}{2 \cdot 3} + \frac{1 \cdot 3 \cdot x^5}{2 \cdot 4 \cdot 5} + \frac{1 \cdot 3 \cdot 5 \cdot x^7}{2 \cdot 4 \cdot 6 \cdot 7} + \dots$	0.5; $\sqrt{2}/2$ ; -1; $y = \arcsin x$
7	$S = \frac{x}{1} - \frac{x^3}{3} + \frac{x^5}{5} - \frac{x^7}{7} + \dots$	$1/\sqrt{3}$ ; $-1/\sqrt{3}$ ; 1; $y = \arctg x$
8	$S = 1 + \sum_{n=1}^{\infty} \frac{(n^2 + 1) \cdot x^n}{n! \cdot 2^n}$	2; 20; -15; $y = (z^2 + z + 1)e^x$ , где $z = x/2$
9	$S = \sum_{n=1}^{\infty} \sin(\pi n / 4) \frac{\sqrt{2^n}}{n!} x^n$ *см. чocky	$\pi/4$ ; $\pi$ ; $-5.5\pi$ ; $y = e^x \sin x$
10	$S = x + \sum_{n=1}^{\infty} (-1)^n \frac{2^{2n} \cdot x^{2n+1}}{(2n)!}$	$\pi/6$ ; $\pi$ ; $4\pi$ ; $y = x \cdot \cos 2x$
11	$S = 2 \cdot \left( \frac{x}{1} + \frac{x^3}{3} + \frac{x^5}{5} + \frac{x^7}{7} + \dots \right)$	0.5; 0.9; 0.99; $y = \ln \frac{1+x}{1-x}$
12	$S = e^{-2} \left( 1 + \sum_{n=1}^{\infty} \frac{(x+2)^n}{n!} \right)$	1; 9; -15; $y = e^x$

13	$S = 1 + \frac{2 \cdot x}{1!} + \frac{3 \cdot x^2}{2!} + \frac{4 \cdot x^3}{3!} + \dots$	1; 9; -11; $y = (1+x) \cdot e^x$
14	$S = \frac{2 \cdot x}{1} - \frac{2^2 x^2}{2} + \frac{2^3 \cdot x^3}{3} - \dots$	1/4; -1/3; 1/2; $y = \ln(1+2x)$
15	$S = 3x + 8x^2 + \dots + n(n+2)x^n + \dots$	0.2; 0.6; 0.9; $y = \frac{x(3-x)}{(1-x)^3}$
16	$S = 1 + \frac{x \ln 2}{1!} + \frac{x^2 \ln^2 2}{2!} + \frac{x^3 \ln^3 2}{3!} + \dots$	2; 10; -10; $y = 2^x$
17	$S = 1 - \frac{2^2 x^2}{2!} + \frac{2^4 x^4}{4!} - \frac{2^6 x^6}{6!} + \dots$	$\pi/6$ ; $\pi$ ; $5\pi$ ; $y = \cos 2x$
18	$S = 1 + \sum_{n=1}^{\infty} \frac{\cos(n\pi/4)}{n!} x^n$ *см. чокку	$\pi/6$ ; $\pi$ ; $-5\pi$ ; $y = e^z \cos z$ где $z = x/\sqrt{2}$
19	$S = 1 - 2x + 3x^2 - 4x^3 + \dots$	0.2; 0.6; 0.9; $y = 1/(1+x)^2$
20	$S = \sum_{n=1}^{\infty} x^n \cdot \sin(n\pi/4)$ *см. чокку	0.5; 0.8; 0.99; $y = (x/\sqrt{2})/(1-\sqrt{2} \cdot x + x^2)$
21	$S = 1 + \frac{\cos(x)}{1!} + \frac{\cos(2x)}{2!} + \dots + \frac{\cos(nx)}{n!} + \dots$ *см. чокку	$\pi/6$ ; $-\pi$ ; $10\pi$ ; $y = e^{\cos x} \cos(\sin x)$
22	$S = -\frac{(2x)^2}{2!} + \frac{(2x)^4}{4!} - \dots (-1)^n \frac{(2x)^{2n}}{(2n)!} + \dots$	$\pi/6$ ; $13\pi/6$ ; $25\pi/6$ ; $y = 2(\cos^2 x - 1)$
23	$S = \sum_{n=0}^{\infty} \frac{1}{2n+1} \cdot \left( \frac{x-1}{x+1} \right)^{2n+1}$	0.5; 0.1; 0.01; $y = 0.5 \cdot \ln(x)$
24	$S = \frac{1}{\sqrt{2}} + \sum_{n=1}^{\infty} \cos\left(\frac{2n+1}{4} \cdot \pi\right) \cdot \frac{(x-\pi/2)^n}{n! \cdot 2^n}$	$2\pi/3$ ; $8\pi/3$ ; $28\pi/3$ ; $y = \cos(x/2)$
25	$S = x + \sum_{n=1}^{\infty} (-1)^n \frac{1 \cdot 3 \cdot 5 \cdot \dots \cdot (2n-1) \cdot x^{2n+1}}{2^n \cdot n! \cdot (2n+1)}$	0.5; 0.9; 1; $y = \ln(x + \sqrt{1+x^2})$
26	$S = x + 2 \cdot \left( \frac{x^3}{1 \cdot 3} - \frac{x^5}{3 \cdot 5} + \frac{x^7}{5 \cdot 7} - \frac{x^9}{7 \cdot 9} + \dots \right)$	$1/\sqrt{3}$ ; $-1/\sqrt{3}$ ; 1 $y = (1+x^2) \cdot \arctg(x)$ ;

27	$S = \frac{\sqrt{3}}{2} + \sum_{n=1}^{\infty} \sin(\pi/3 + n\pi/2) \cdot \frac{\pi^n (x-1)^n}{3^n \cdot n!}$	0.5; 6.5; 13.5; $y = \sin(\pi x/3)$
28	$S = \sum_{n=1}^{\infty} (-1)^{n-1} (1+2^n) \frac{x^n}{n}$	0.25; 0.45; 0.5; $y = \ln(1+3x+2x^2)$
29	$S = -1 + \frac{x+1}{3} + \sum_{n=2}^{\infty} \frac{2 \cdot 5 \cdot 8 \cdot \dots \cdot (3n-4)}{3^n n!} (x+1)^n$	-1.331; -0.729; 0; $y = \sqrt[3]{x}$
30	$S = 4 + \sum_{n=1}^{\infty} (2+2^n) \frac{x^n}{n!}$	2; -1; -10; $y = (1+e^x)^2$

\*) для рядов, отмеченных звёздочкой, при оценке погрешности в членах ряда не учитывать синусы и косинусы.



## ЛАБОРАТОРНАЯ РАБОТА № 5

### Одномерные массивы

#### Краткие теоретические сведения

**Массивом** называется последовательная группа переменных одного типа. Массивы служат для размещения набора данных, которые необходимо сохранить и использовать в процессе выполнения программы.

Элементы массива имеют одно и то же имя, но различаются порядковым номером (индексом), что позволяет компактно записывать множество операций с помощью циклов. В языке C#, как и во многих других языках, индексы задаются целочисленным типом.

Число индексов характеризует размерность массива. Каждый индекс изменяется в некотором диапазоне  $[a, b]$ , который называется граничной парой, где  $a$  – нижняя граница, а  $b$  – верхняя граница индекса. При объявлении массива границы задаются выражениями. Если все границы заданы константными выражениями, то число элементов массива известно в момент его объявления и ему может быть выделена память еще на этапе трансляции. Такие массивы называются статическими. Если же выражения, задающие границы, зависят от переменных, то такие массивы называются динамическими.

Язык C# поддерживает два вида или две категории типов: типы значений (value types) и типы ссылок (reference types). Элементами массива могут быть как значения, так и ссылки. Массив значимых типов хранит значения, массив ссылочных типов – ссылки на элементы. Всем элементам при создании массива присваиваются значения по умолчанию: нули для значимых типов и null – для

ссылочных. Массивы ссылочного типа являются массивами динамическими и память им отводится динамически в области памяти, называемой «кучей» (heap).

Массивами в С# можно пользоваться практически так же, как и в других языках программирования. Тем не менее у них имеется одна особенность: они реализованы в виде объектов. Реализация массивов в виде объектов дает ряд существенных преимуществ, и далеко не самым последним среди них является возможность утилизировать неиспользуемые массивы посредством "сборки мусора".

Поскольку в С# массивы реализованы в виде объектов, то для того чтобы воспользоваться массивом в программе, требуется двух-этапная процедура. Во-первых, необходимо объявить переменную, которая может обращаться к массиву:

```
тип[] имя_массива;
```

во-вторых, нужно создать экземпляр массива, используя оператор new:

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

где **тип** объявляет конкретный тип элемента массива, а **размер** определяет число элементов массива. Тип элемента определяет тип данных каждого элемента, составляющего массив. Квадратные скобки указывают на то, что объявляется одномерный массив.

Здесь необходимо отметить, что в отличии от других языков программирования (С, С++, Fortran или VBA), квадратные (или круглые) скобки следуют после названия типа, а не имени массива.

Рассмотрим конкретный пример. В приведенной ниже строке кода создается массив типа `int` из десяти элементов, и переменная `array`, которая является ссылкой на массив типа `int[]` с элементами типа `int`.

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

В переменной `array` хранится ссылка на область памяти, выделяемой для массива оператором `new`. Эта область памяти должна быть достаточно большой, чтобы в ней могли храниться десять элементов массива типа `int`.

Приведенное выше объявление массива можно разделить на два отдельных оператора. Например:

```
int[] array;           // объявление массива с
                        // именем array
array = new int[10];    // резервирование памяти для
                        // 10 чисел типа int
```

Доступ к отдельному элементу массива осуществляется по индексу. В языке C# индекс первого элемента всех массивов является нулевым. В частности, массив `array` состоит из 10 элементов с индексами от 0 до 9. Для индексирования массива достаточно указать номер требуемого элемента в квадратных скобках. Так, первый элемент массива `array` обозначается как `array[0]`, а последний его элемент – как `array[9]`. Ниже приведен пример программы, в которой заполняются все элементы массива `iArray` и массива `chArray`.

### **Пример 1.**

```
// Заполнение массивов
using System;
namespace Example5
{
    class Example5_1
    {
```

```

static void Main()
{
    int j;
    Console.WriteLine("\n\n Одномерный
массив iArray");
    int[] iArray = new int[10];
    for (j = 0; j < 10; j++)
        iArray[j] = j * j;
    // присваивание значений
    // элементам в цикле
    for (j = 0; j < 10; j++)
        // вывод элементов
        Console.WriteLine("\n " + j + " "
        + iArray[j]);
    Console.WriteLine("\n Одномерный
массив chArray с инициализацией");
    char[] chArray =
    { 'a', 'b', 'c', 'd' };
    // Объявление с инициализацией
    j = -1;
    do
    {
        j++;
        Console.WriteLine("\n " +
        j + " " + chArray[j]);
    }
    while (chArray[j] != 'd');
}

```

```

        // вывод элементов массива
        Console.WriteLine();
        Console.WriteLine("\n Значения
        присвоены ");
        Console.WriteLine("не всем элемента
        массива iiArray \n");
        int[] iiArray = new int[10];
        for (j = 0; j < 6; j++)
            iiArray[j] = j * j;
        iiArray[9] = 81;
        foreach (int jj in iiArray)
        { Console.WriteLine(" " + jj); }
        Console.WriteLine("\n\n");
        Console.WriteLine(" ");
    }
}
}

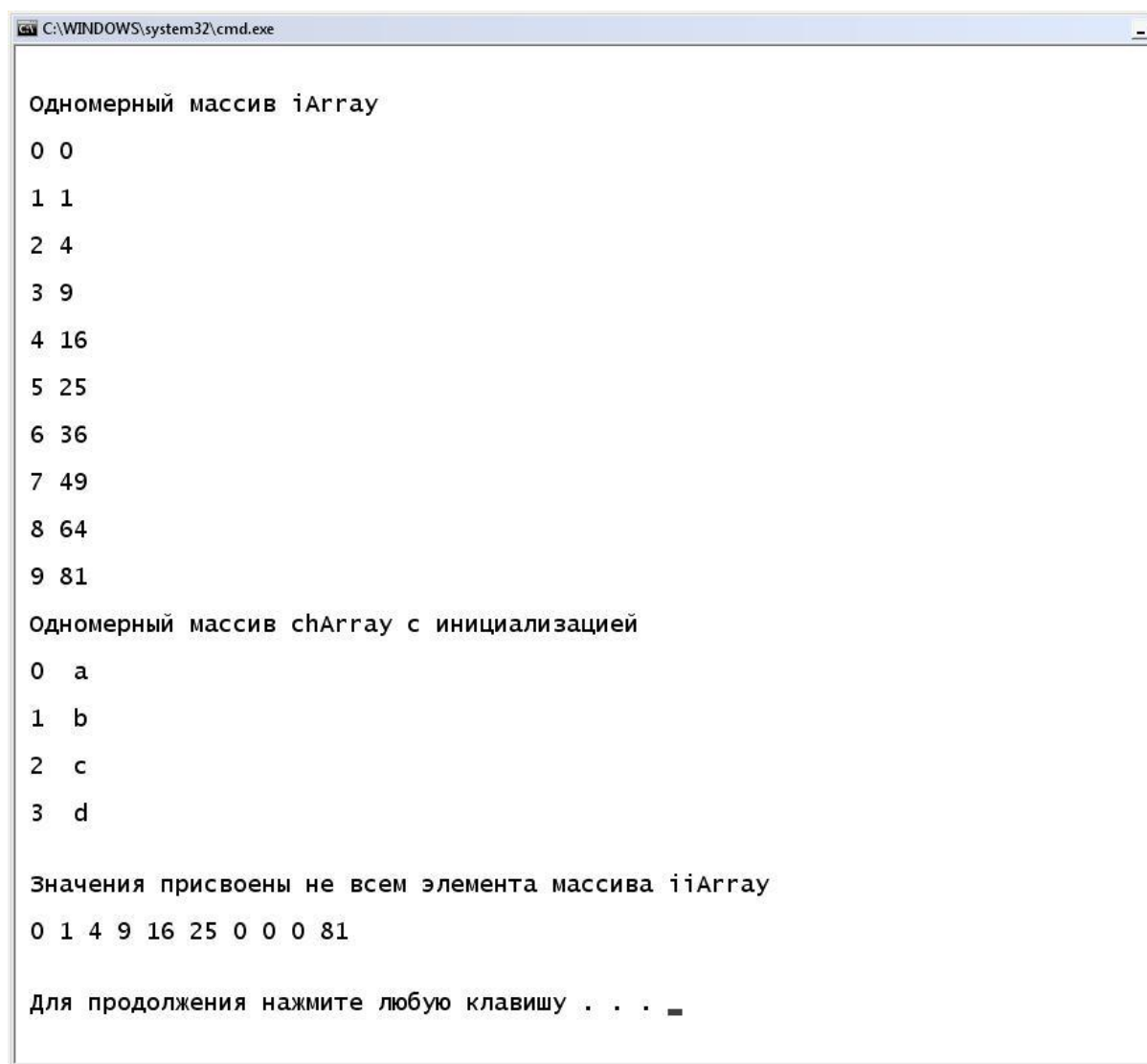
```

В начале программы объявлен массив `iArray` из 8 целых чисел. Потом в цикле присваиваются значения элементам. Аналогичный цикл используется и для вывода значений элементов на экран.

Далее объявляется массив символов `chArray` без указания количества элементов с инициализацией, после чего поэлементно выводится в цикле `do-while`.

Кроме описанных ранее, в C# определен цикл `foreach`. Он перебирает подряд все элементы массива. В программе `foreach` применяется к массиву `chArray`. Выражение `foreach(char jj in iiArray){...}`

показывает, что все элементы массива `iiArray` поочередно присваиваются переменной цикла `char`, тип которой должен соответствовать типу массива. На местах элементов, которым не присвоены значения, цикл `foreach` выводит нули, что демонстрируется на примере массива `iiArray`.



```
C:\WINDOWS\system32\cmd.exe

Одномерный массив iiArray
0 0
1 1
2 4
3 9
4 16
5 25
6 36
7 49
8 64
9 81
Одномерный массив chArray с инициализацией
0 a
1 b
2 c
3 d

Значения присвоены не всем элемента массива iiArray
0 1 4 9 16 25 0 0 0 81

Для продолжения нажмите любую клавишу . . .
```

Следует, однако, иметь в виду, что переменная цикла в операторе `foreach` служит только для вывода. Это означает, что, прис-

ваивая этой переменной новое значение, нельзя изменить содержимое массива.

### **Инициализация массивов**

Как уже отмечалось, массив – это структура данных, содержащая несколько элементов одного типа. В следующих примерах показано создание и инициализация одномерных массивов.

```
// Объявление массива
int[ ] array1 = new int[5];
// Объявление и инициализация массива
int[ ] array2 = new int[ ] {1, 3, 5, 7, 9} ;
// Альтернативный синтаксис
int[ ] array3 = {1, 2, 3, 4, 5, 6} ;
// при инициализации массива его размер можно
// указывать явным образом, но этот размер
// должен совпадать с числом инициализаторов
int[ ] array4 = new int[10]
{99, 10, 100, 18, 1, 78, 22, 69};
```

### **Ввод-вывод массивов**

Заполнить массив, т. е. определить значения элементов массива можно следующими способами:

1. при помощи оператора присваивания;
2. непосредственным вводом с клавиатуры;
3. подготовкой и вводом данных из текстового файла;
4. использования датчика случайных чисел;
5. заполнением массива при помощи стандартных функций

## Пример 2.

// Ввод массива с клавиатуры

using System;

namespace Example5

{

class Example5\_2

{

static void Main()

{

int j;

// начальное значение

string strValue;

int[] iArray = new int[10];

for (j = 0; j < 10; j++)

{

strValue = Console.ReadLine();

// ввод и присваивание значений

iArray[j] = Convert.ToInt32(strValue);

}

for (j = 0; j < 10; j++)

// вывод элементов

Console.WriteLine("\n " + j + " " +  
iArray[j]);

}

}

}

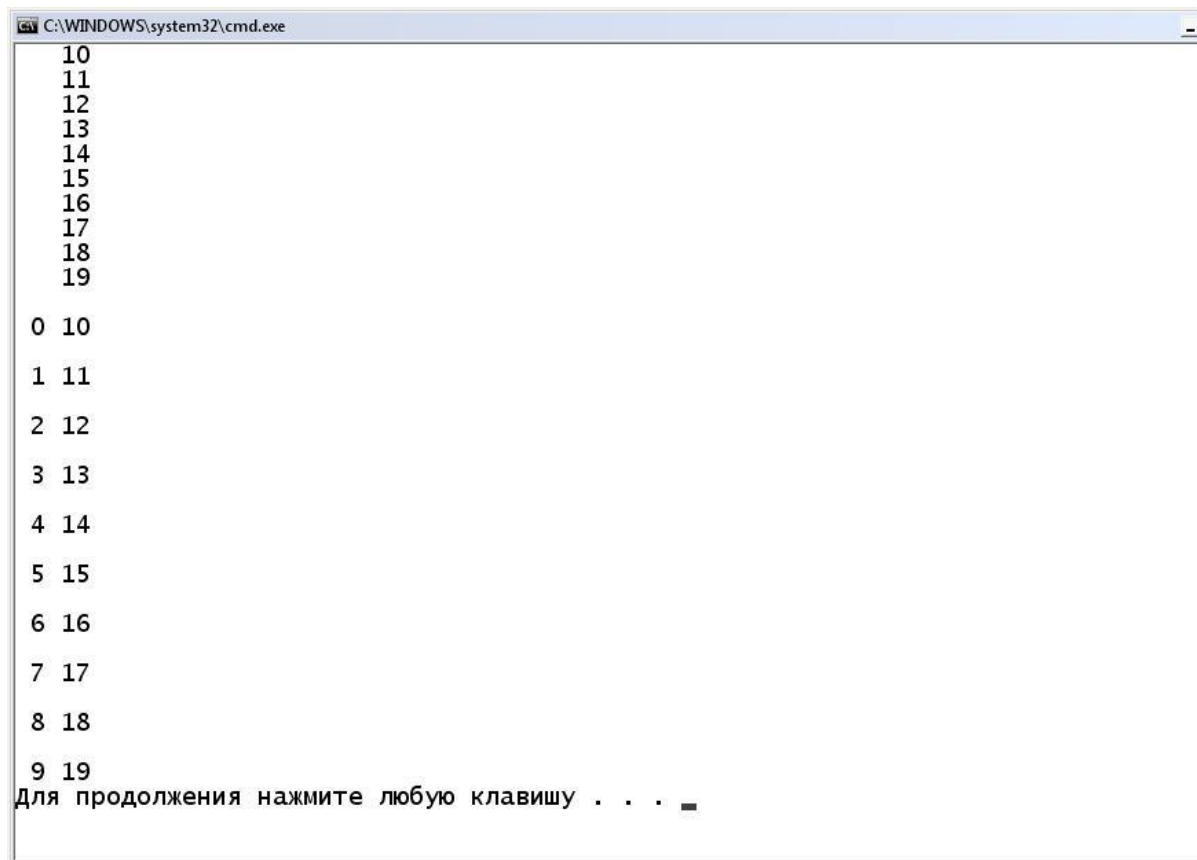


Для организации ввода необходимо объявить строковую переменную, которой присваивается очередное введенное с клавиатуры число. Следующий оператор

```
i Array[j] = Convert.ToInt32(strValue);
```

преобразует строковую переменную `strValue` в целое 32-разрядное число. Ввод и преобразование происходит в цикле, после завершения которого массив `i Array` содержит исходные данные.

Каждый объект (переменная), каждый операнд при вычислении выражения, в том числе и само выражение характеризуется парой (значение, тип), задающей значение выражения и его тип. В процессе вычислений зачастую возникает необходимость преобразования типов – необходимость преобразовать пару (значение1, тип1) к паре (значение2, тип2). Исходная пара называется ис-



```
C:\WINDOWS\system32\cmd.exe
10
11
12
13
14
15
16
17
18
19
0 10
1 11
2 12
3 13
4 14
5 15
6 16
7 17
8 18
9 19
Для продолжения нажмите любую клавишу . . .
```

точником преобразования, заключительная – целью преобразования. Некоторые преобразования типов выполняются автоматически. Такие преобразования называются неявными, и они часто встречаются при вычислении выражений. Все остальные преобразования называются явными и для них существуют разные способы таких преобразований – операция кастинга (приведение к типу), методы специального класса `Convert`, специальные методы `ToString`, `Parse`.

Все скалярные типы (арифметический, логический, символьный) имеют статический метод `Parse`, аргументом которого является строка, а возвращаемым результатом – объект соответствующего типа. Метод явно выполняет преобразование текстового представления в тот тип данных, который был целью вызова статического метода.

Для преобразований внутри арифметического типа можно использовать кастинг – приведение типа. Для преобразований строкового типа в скалярный тип можно применять метод `Parse`, а в обратную сторону – метод `ToString`.

Однако, во всех ситуациях, когда требуется выполнить преобразование из одного базового встроенного типа в другой базовый тип, можно использовать методы универсального класса `Convert`, встроенного в пространство имен `System`.

Методы класса `Convert` поддерживают общий способ выполнения преобразований между типами. Класс `Convert` содержит 15 статических методов вида (`ToInt16()`, `ToInt32()`, `ToInt64()`, ..., `ToDouble`, `ToDecimal`, ..., ). Единственным исключением является тип `object` – метода `ToObject` нет по понятным причинам, поскольку для всех типов существует неявное преобра-

зование к типу `object`. Каждый из этих 15 методов перегружен, и его аргумент может принадлежать к любому из упомянутых типов. С учетом перегрузки с помощью методов этого класса можно осуществить любое из возможных преобразований одного типа в другой.

### **Пример 3.**

```
// Заполнение массива с помощью
// генератора случайных чисел
using System;
namespace Example5
{
    class Example5_3
    {
        static void Main()
        {
            int j, num1, num2;
            string str;
            double db1, db2;
            Random rnd = new Random();
            int[] iArray1 = new int[10];
            int[] iArray2 = new int[10];
            double[] dArray1 = new double[10];
            double[] dArray2 = new double[10];
            for (j = 0; j < 10; j++)
            {
                iArray1[j] = rnd.Next(1, 101);
                iArray2[j] = 50 - rnd.Next(1,
```

```

        101);
    }
    for (j = 0; j < 10; j++)
    {
        num1 = rnd.Next(1, 101);
        db1 = Convert.ToDouble(num1);
        dArray1[j] = db1 +
        Convert.ToDouble(rnd.Next(1,
        101)) / 100;
        num2 = 50 - rnd.Next(1, 101);
        db2 = Convert.ToDouble(num2);
        dArray2[j] = db2 -
        Convert.ToDouble(rnd.Next(1,
        101)) / 100;
    }
    Console.WriteLine("\n -----
    -----");
    Console.WriteLine("\n    Массивы типа int
    Массивы типа double");
    Console.WriteLine("\n -----
    -----");
    for (j = 0; j < 10; j++)
    {
        str = string.Format("\n {0, 4:D}
        {1, 6:D} {2, 6:D} {3, 8:D}
        {4, 8:F2} {5, 8:F2}",
        j, iArray1[j],

```

```

        i Array2[j ], j , dArray1[j ],
        dArray2[j ]));
        Console.WriteLine(str);
    }
    Console.WriteLine("\n -----
    -----");
    Console.WriteLine();
}
}
}

```

```

C:\WINDOWS\system32\cmd.exe
-----
Массивы типа int      Массивы типа double
-----
0      41      26      0      12,96      -8,33
1      84      -27     1      4,78      0,16
2       8       0     2      4,06     -14,56
3      33     -12     3     50,16     28,13
4      58     -35     4     25,12      0,64
5       6      12     5     31,63     25,65
6      31      -2     6      3,07     -31,31
7      76     -37     7     92,49     43,90
8      41     -26     8     47,13      3,54
9      96      -4     9     67,51     -24,30
-----
Для продолжения нажмите любую клавишу . . .

```

В данном примере для заполнения массива используется генератор случайных чисел. Для генерирования последовательного ряда

случайных чисел служит класс `Random`. Такие последовательности чисел оказываются полезными в самых разных ситуациях включая имитационное моделирование. Начало последовательности случайных чисел определяется некоторым начальным числом, которое может задаваться автоматически или указываться явным образом.

В классе `Random` определяются два конструктора:

```
public Random ()  
public Random(int seed)
```

Первый конструктор создает объект типа `Random`, использующий системное время определения начального числа. А во втором конструкторе используется начальное значение `seed`, задаваемое явным образом.

В первом цикле заполняются массивы `iArray1` и `iArray2`, причем массив `iArray1` заполняется числами от 0 до 100, массив `iArray2` заполняется числами от -50 до 50. В этих же интервалах находятся и числа `num1` и `num2`, которые в следующих строках преобразуются к типу `double`. Оператор `Convert.ToDouble(rnd.Next(1, 101)) / 100;` генерирует случайные числа, находящиеся в интервале от 0.0 до 1.0. Таким образом, массивы `dArray1` и `dArray2` будут содержать числа типа `double`. Основные методы класса `Random` представлены в таблице:

Метод	Назначение
<code>Public virtual int Next(int upperBound)</code>	Возвращает следующее случайное целое число, которое будет находиться в пределах от 0 до

	<i>upperBound</i> -1 включительно
Public virtual int Next (int <i>lowerBound</i> , int <i>upperBound</i> )	Возвращает следующее случайное целое число, которое будет находи- ться в пределах от <i>lowerBound</i> до <i>upperBound</i> -1 включительно
Public virtual double NextDouble (int <i>upperBound</i> )	Возвращает следующее случайное число с плавающей точкой, больше или равно 0, 0 и меньше 1, 0

#### Пример 4.

```
// Ввод массива в файл и
// вывод массива из файла
using System;
using System.IO;
namespace Example5
{
    class Example5_4
    {
        static void Main()
        {
            int j;
            string strValue;
            int[] iArray1 = new int[10];
            int[] iArray2 = new int[10];
            StreamReader sRead = new
            StreamReader("C: \\\dat.txt");
            StreamWriter sWrite = new
```

```

        StreamWriter("C: \\res. txt");
        for (j = 0; j < 10; j++)
        {
            strValue = sRead. ReadLi ne();
            i Array1[j] =
            Convert. ToInt32(strValue);
            i Array2[j] = 10 * i Array1[j];
            strValue = string. Format("\n {0,
            4: D} {1, 6: D} {2, 6: D}",
            j, i Array1[j], i Array2[j]);
            Console. Wri teLi ne(strValue);
            Console. Wri teLi ne();
            sWri te. Wri teLi ne(i Array2[j]);
        }
        sRead. Cl ose();
        sWri te. Cl ose();
    }
}
}

```

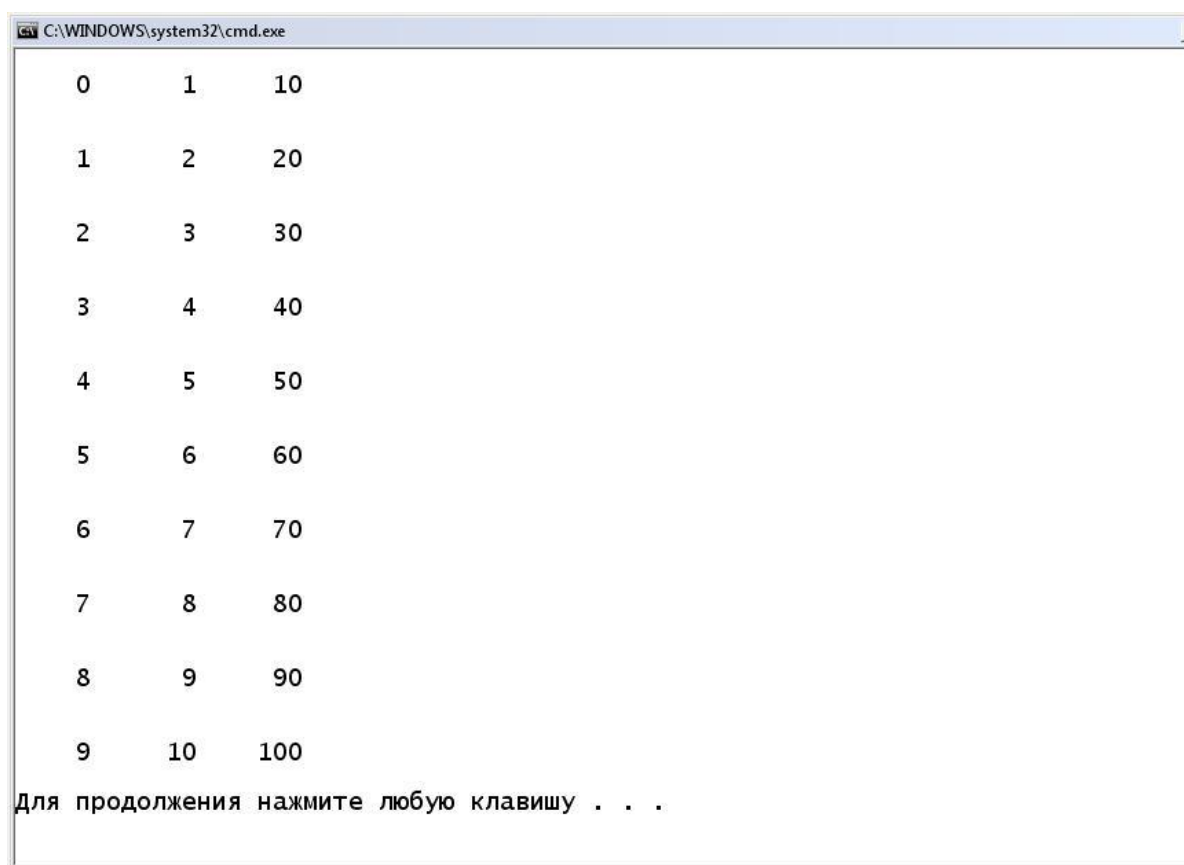
В этом примере происходит считывание из файла и запись в файл. Имя файла включает полный путь и, следовательно, содержит служебный знак \ (обратный слеш). Если используемые символы являются служебными, то на экран они не выводятся. Однако, в случае необходимости использовать эти символы в выводимом тексте, следует перед каждым таким символом поставить дополнительный символ \.



Все действия, т. е. считывание из файла и запись в файл, а также вывод на экран организованы в одном цикле. В переменной `strValue` формируется текстовая строка вывода на экран, содержащая форматы, заключенные в фигурные скобки

```
"\n {0, 4: D} {1, 6: D} {2, 6: D}",
```

где в первой позиции расположены цифры 0, 1 и 2, которые являются соответственно указателями на `j`, `iArray1`, `iArray2`. Код форматирования `4: D` определяет число позиций (4), отведенных в строке вывода для данного целого числа (`D` – формат типа `decimal`).



```
C:\WINDOWS\system32\cmd.exe

0      1      10
1      2      20
2      3      30
3      4      40
4      5      50
5      6      60
6      7      70
7      8      80
8      9      90
9      10     100

Для продолжения нажмите любую клавишу . . .
```

### Пример 5.

// Вычислить сумму и среднеарифметическое всех

```

// элементов одномерного массива,
// состоящего из 15 элементов.
// Заполнение массива происходит при
// помощи генератора случайных чисел
using System;
namespace Example5
{
    class Example5_5
    {
        static void Main()
        {
            int j, num;
            string str;
            string str1 = "Сумма", str2 =
                "Сумма";
            string str3 = "Среднее", str4 =
                "Среднее";
            double db1, db2;
            double sum1 = 0, sum2 = 0,
                sum3 = 0, sum4 = 0;
            Random rnd = new Random();
            int[] iArray1 = new int[15];
            int[] iArray2 = new int[15];
            double[] dArray1 = new double[15];
            double[] dArray2 = new double[15];
            for (j = 0; j < 15; j++)
            {

```

```

        iArray1[j] = rnd.Next(1, 101);
        iArray2[j] = 50 - rnd.Next(1,
            101);
        sum1 += iArray1[j];
        sum2 += iArray2[j];
    }
    for (j = 0; j < 15; j++)
    {
        num = rnd.Next(1, 101);
        db1 = Convert.ToDouble(num);
        dArray1[j] = db1 +
            Convert.ToDouble(rnd.Next(1,
                101)) / 100;
        num = 50 - rnd.Next(1, 101);
        db2 = Convert.ToDouble(num);
        dArray2[j] = db2 -
            Convert.ToDouble(rnd.Next(1,
                101)) / 100;
        sum3 += dArray1[j];
        sum4 += dArray2[j];
    }
    Console.WriteLine("\n -----
    -----
    -----");
    Console.WriteLine("\n
    Массивы типа int                Массивы
    типа double");

```

```

Console.WriteLine("\n -----
-----
-----");
for (j = 0; j < 15; j++)
{
    str = string.Format("\n {0, 10:D}
    {1, 10:D} {2, 10:D} {3, 10:D}
    {4, 10:F2} {5, 10:F2}",
    j, iArray1[j], iArray2[j],
    j, dArray1[j], dArray2[j]);
    Console.WriteLine(str);
}
Console.WriteLine("\n -----
-----
-----");
Console.WriteLine("\n {0, 10} {1, 10}
{2, 10} {3, 10}
{4, 10:F2} {5, 10:F2}",
str1, sum1, sum2,
str2, sum3, sum4);
Console.WriteLine("\n {0, 10}
{1, 10:F2}
{2, 10:F2} {3, 10}
{4, 10:F2} {5, 10:F2}",
str3, sum1 / 15, sum2 /
15, str4, sum3 /
15, sum4 / 15);

```

Console.WriteLine();

}

}

}

C:\WINDOWS\system32\cmd.exe

-----					
Массивы типа int			Массивы типа decimal		
-----					
0	86	13	0	3,53	3,68
1	86	6	1	88,49	-12,13
2	53	20	2	40,27	-48,47
3	92	29	3	75,48	-49,61
4	69	-38	4	48,65	10,96
5	71	47	5	71,92	-50,40
6	100	-21	6	72,20	-49,98
7	3	2	7	13,22	-23,71
8	81	-5	8	9,82	-35,19
9	55	-35	9	58,68	-34,55
10	21	-17	10	86,69	33,72
11	38	-39	11	88,74	-43,91
12	31	19	12	91,70	-34,92
13	57	29	13	82,09	32,88
14	13	-38	14	53,06	-11,68
-----					
Сумма	856	-28	Сумма	884,54	-313,31
СрАрф	57,07	-1,87	СрАрф	58,97	-20,89

Для продолжения нажмите любую клавишу . . . █

Массивы типа `int` `iArray1`, `iArray2`, а также типа `double` `dArray1`, `dArray2` заполняются генератором случайных чисел. В первом цикле случайными числами инициализируется массивы це-

лого типа и одновременно подсчитывается сумма элементов обоих массивов – sum1 и sum2. Переменная этого цикла одновременно является индексом массива. На каждом шаге цикла к сумме элементов каждого массива добавляется очередной элемент. По окончании цикла переменные sum1 и sum2 будут содержать полную сумму всех элементов массивов iArray1, iArray2.

В втором цикле инициализируются массивы чисел с плавающей точкой и одновременно по аналогичной схеме подсчитывается сумма элементов массивов dArray1, dArray2 – sum3 и sum4.

Далее формируется заголовок таблицы вывода, которая заполняется в третьем цикле. Для массивов типа int использован формат D, а для массивов типа double – формат F (fixed point).

### **Пример 6.**

```
// Вычислить сумму всех четных элементов
// одномерного массива, состоящих из 10 элементов
// Заполнение одномерного массива происходит при
// помощи генератора случайных чисел
using System;
namespace Example5
{
    class Example5_6
    {
        static void Main()
        {
            int j, num, sum = 0;
            Random rnd = new Random();
            int[] iArray = new int[10];
```

```

for (j = 0; j < 10; j++)
{
    iArray[j] = rnd.Next(1, 101);
}
for (j = 0; j < 10; j++)
{
    num = Convert.ToInt32(iArray[j]
    % 2);
    if (num == 0) sum += iArray[j];
}
foreach (int jj in iArray)
{ Console.WriteLine(" " + jj); }
Console.WriteLine("\n\n");
Console.WriteLine("\n Сумма четных
элементов = " + sum);
Console.WriteLine();
Console.WriteLine(" ");
}
}
}

```

Отличие данного примера от предыдущего заключается в том, что требуется подсчитать сумму не всех элементов массива, а только четных

```

num = Convert.ToInt32(iArray[j] % 2);
if (num == 0) sum += iArray[j];

```

Сначала определяется остаток от деления элемента массива на 2 с помощью выражения `iArray[j] % 2`, результат которого прео-

бразуется к целому типу. Если выполняется условие `num == 0`, то значение элемента `iArray[j]` является величиной четной и происходит суммирование данного элемента с переменной `Sum`.



```
C:\WINDOWS\system32\cmd.exe
90 31 16 67 24 98 92 88 98 77

Сумма четных элементов = 506
Для продолжения нажмите любую клавишу . . . _
```

### Пример 7.

```
// Определить индексы второго положительного
// и третьего отрицательного элементов
// одномерного массива, состоящих из 20 элементов
// Заполнение массива происходит
// при помощи генератора случайных чисел
using System;
```

```
namespace Example5
```

```
{
```

```
    class Example5_7
```

```
    {
```

```
        static void Main()
```

```
        {
```

```
            int j, jnum = 0;
```

```
            Random rnd = new Random();
```

```
            int[] iArray = new int[20];
```

```
            for (j = 0; j < 20; j++)
```

```
            {
```

```
                iArray[j] = 50 - rnd.Next(1,
```



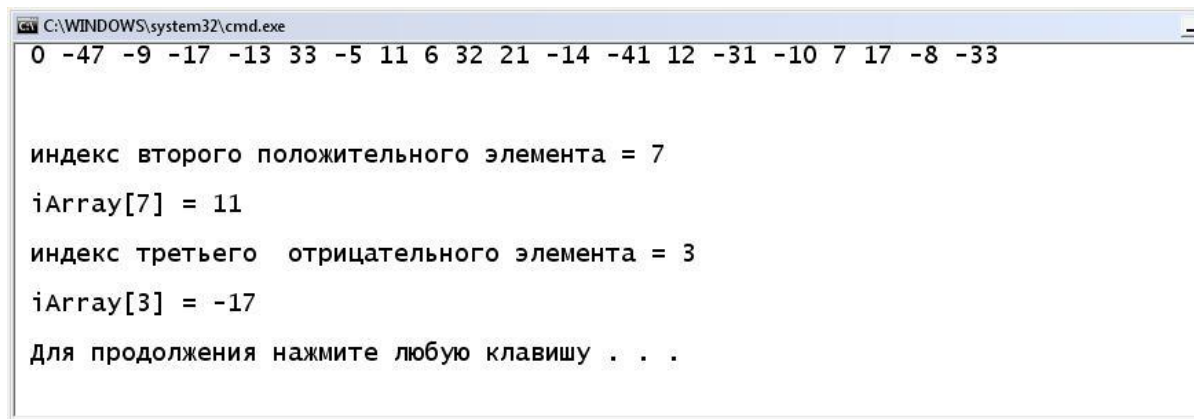
```

        101);
    }
    for (j = 0; j < 20; j++)
    {
        if (iArray[j] > 0) jnum += 1;
        if (jnum == 2) break;
    }
    foreach (int jj in iArray)
    { Console.WriteLine(" " + jj); }
    Console.WriteLine("\n\n");
    Console.WriteLine("\n индекс второго
    положительного элемента = " + j);
    Console.WriteLine("\n iArray[" +
    j + "] = " + iArray[j]);
    jnum = 0;
    for (j = 0; j < 20; j++)
    {
        if (iArray[j] < 0) jnum += 1;
        if (jnum == 3) break;
    }
    Console.WriteLine("\n индекс третьего
    отрицательного элемента = " + j);
    Console.WriteLine("\n iArray[" +
    j + "] = " + iArray[j]);
    Console.WriteLine();
    Console.WriteLine(" ");
}

```

```
}  
}
```

Поиск второго положительного элемента массива `iArray` происходит в цикле. Переменная `j num` определяет количество положительных элементов, которые встретились при выполнении цикла. Как только значение выражения `j num == 2` будет равно `true` оператор `break` прерывает выполнение цикла.



```
C:\WINDOWS\system32\cmd.exe  
0 -47 -9 -17 -13 33 -5 11 6 32 21 -14 -41 12 -31 -10 7 17 -8 -33  
  
индекс второго положительного элемента = 7  
iArray[7] = 11  
индекс третьего отрицательного элемента = 3  
iArray[3] = -17  
Для продолжения нажмите любую клавишу . . .
```

Аналогичный алгоритм используется при нахождении третьего отрицательного элемента.

### **Пример 8.**

```
// Задан одномерный массив размером N.  
// Сформировать два массива, включив  
// в первый - четные элементы исходного  
// массива, а во второй - нечетные элементы.  
// Отсортировать массивы в порядке возрастания.  
// Заполнение массива происходит при помощи  
// генератора случайных чисел
```

```

using System;
namespace Example5
{
    class Example5_8
    {
        static void Main()
        {
            int jnum = 0, N = 20;
            int jAA = 0, jBB = 0;
            int j, k, temp;
            Random rnd = new Random();
            int[] iArray = new int[N];
            int[] jA = new int[N];
            int[] jB = new int[N];
            for (j = 0; j < N; j++)
            {
                iArray[j] = rnd.Next(1, 101);
            }
            Console.WriteLine("\n исходный массив\n\n");
            foreach (int jj in iArray)
            { Console.WriteLine(" " + jj); }
            Console.WriteLine("\n\n");
            for (j = 0; j < N; j++)

```

```

{
    j num = i Array[j ] / 2;
    i Array[j ] =
    Convert.ToInt32(i Array[j ]);
    if (i Array[j ] == j num * 2)
    {
        j A[j AA] = i Array[j ];
        j AA += 1;
    }
    else
    {
        j B[j BB] = i Array[j ];
        j BB += 1;
    }
}

Console.WriteLine("\n массив A[]
\n\n");
foreach (int jj in jA)
{ Console.WriteLine(" " + jj); }
Console.WriteLine("\n\n");
Console.WriteLine("\n массив B[]
\n\n");
foreach (int jj in jB)
{ Console.WriteLine(" " + jj); }

```

```

Console.WriteLine("\n\n");
// Сортировка массива A
jAA -= 1;
for (k = 0; k < jAA; k++)
{
    for (j = 0; j < jAA; j++)
    {
        if (jA[j + 1] < jA[j])
        {
            temp = jA[j];
            jA[j] = jA[j + 1];
            jA[j + 1] = temp;
        }
    }
}

// Сортировка массива B
jBB -= 1;
for (k = 0; k < jBB; k++)
{
    for (j = 0; j < jBB; j++)
    {
        if (jB[j + 1] < jB[j])
        {
            temp = jB[j];

```

```

        jB[j] = jB[j + 1];
        jB[j + 1] = temp;
    }
}
}
Console.WriteLine("\n отсортированный
массив A[] \n\n");
foreach (int jj in jA)
{ Console.WriteLine(" " + jj); }
Console.WriteLine("\n\n");
Console.WriteLine("\n отсортированный
массив B[] \n\n");
foreach (int jj in jB)
{ Console.WriteLine(" " + jj); }
Console.WriteLine("\n\n");
}
}
}

```

Для определения четности или нечетности очередного элемента массива. Если результат данного преобразования равен исходному значению, то данный элемент массива содержит четное число, В противном случае – число нечетное. По окончании данной процедуры значения всех элементов массива jA – четные числа, массива jA – нечетные числа.

Далее происходит сортировка элементов массивов  $jA$  и  $jB$  по возрастанию. Процесс сортировки происходит следующим образом: если элемент с индексом  $j$  больше элемента с индексом  $j+1$ , то выполняется процедура перестановки

```
C:\WINDOWS\system32\cmd.exe

исходный массив

24 96 28 67 12 50 91 11 93 61 87 60 60 15 79 86 45 51 48 13

массив A[]

24 96 28 12 50 60 60 86 48 0 0 0 0 0 0 0 0 0 0 0

массив B[]

67 91 11 93 61 87 15 79 45 51 13 0 0 0 0 0 0 0 0 0

отсортированный массив A[]

12 24 28 48 50 60 60 86 96 0 0 0 0 0 0 0 0 0 0 0

отсортированный массив B[]

11 13 15 45 51 61 67 79 87 91 93 0 0 0 0 0 0 0 0 0

Для продолжения нажмите любую клавишу . . .
```

```
temp = j B[j];
j B[j] = j B[j + 1];
j B[j + 1] = temp;
```

и завершении цикла массивы  $jA$  и  $jB$  будут отсортированы по возрастанию.

## 2. Практическая часть

### Задание к лабораторной работе

Для заданного условия составить процедуру и придумать несколько наборов тестовых данных для отладки. Возможно использование как статических массивов, так и динамических. Ввод исходных данных осуществить из файла или с клавиатуры.

1. Задан одномерный массив размером  $N$ . Определить количество отрицательных чисел, количество положительных чисел и среднее арифметическое всех чисел.
2. Задан одномерный массив размером  $N$ . Сформировать два массива размером  $N/2$ , включая в первый элементы исходного массива с четными индексами, а во второй – с нечетными. Вычислить суммы элементов каждого из массивов.
3. Определить среднее арифметическое значение первого отрицательного и последнего положительного элементов одномерного массива, размер которого равен  $M$ .
4. Определить число отрицательных элементов, расположенных перед наибольшим положительным элементом одномерного массива, размер которого равен  $M$ .
5. В заданном одномерном массиве размером  $N$  поменять местами первый и последний положительные элементы.
6. Написать программу для вычисления суммы и среднего арифметического значения всех элементов заданного одномерного массива  $A$ , состоящего из 10-ти элементов.
7. Написать программу для вычисления суммы положительных элементов, заданного массива  $A$ , состоящего из 20-ти элементов.



8. Написать программу для вычисления суммы четных элементов заданного массива  $A$ , состоящего из 20-ти элементов.
9. Написать программу для определения количества положительных элементов, заданного массива  $A$ , состоящего из 20-ти элементов.
10. Написать программу для определения индексов положительных элементов, заданного массива  $A$ , состоящего из 20-ти элементов.
11. Написать программу для вычисления среднего арифметического значения всех элементов заданного массива  $D$ . Для отрицательных элементов использовать их абсолютные значения.
12. Написать программу для поиска в заданном массиве  $B$ , состоящем из 10-ти элементов, третьего положительного элемента и его индекса. Известно, что хотя бы один положительный элемент в массиве  $B$  имеется.
13. Написать программу поиска в заданном массиве  $B$ , состоящем из 20-ти элементов, третьего положительного элемента и его индекса.
14. Написать программу для поиска в заданном массиве  $A(15)$  наибольшего значения элемента и его индекса.
15. Написать программу, в которой производится перестановка четных и нечетных элементов, заданного массива  $C$ .
16. Для заданного массива  $A$ , состоящего не более чем из 50-ти элементов, найти наименьший элемент и переставить его со вторым по порядку отрицательным элементом массива.
17. Написать программу по упорядочению элементов заданного массива  $B$  в следующем порядке: сначала идут положительные числа, потом – нули и в конце – отрицательные.

18. Написать программу сортировки по возрастанию заданного массива В, состоящего из 10-ти элементов.
19. Написать программу. Для заданного массива В, состоящего из 10-ти элементов, изменить порядок следования его элементов на обратный.
20. Написать программу, в которой для заданного массива В, состоящего из 10-ти элементов, его элементы перемещались бы например на 7 позиций вправо. При этом 7 элементов из конца массива перемещаются в начало.
21. Задан массив А. Поместить положительные элементы этого массива в массив В, а отрицательные элементы – в массив С.
22. В заданном векторе (одномерном массиве) найти сумму первого и последнего отрицательного элемента
23. В заданном векторе (одномерном массиве) найти: разность первого и последнего нечетного элементов
24. В заданном векторе (одномерном массиве) найти: индексы наименьшего и наибольшего из элементов
25. В заданном векторе (одномерном массиве) найти: произведение трех наименьших элементов вектора
26. В заданном векторе (одномерном массиве) найти: сумму элементов вектора с четными индексами
27. В заданном векторе (одномерном массиве) найти: разность первого положительного и последнего отрицательного элемента
28. В заданном векторе (одномерном массиве) найти: число отрицательных элементов с нечетными индексами
29. В заданном векторе (одномерном массиве) найти: число элементов, расположенных после его наибольшего отрицательного элемента.

30. В заданном векторе (одномерном массиве) найти: наибольший отрицательный и наименьший положительные элементы.

## **ЛАБОРАТОРНАЯ РАБОТА № 6**

### **Многомерные массивы**

#### **Краткие теоретические сведения**

В языке C++ все массивы являются статическими; более того, все массивы являются 0-базируемыми. Это означает, что нижняя граница всех индексов массива фиксирована и равна нулю.

В языке C# снято существенное ограничение языка C++ на статичность массивов. Массивы в языке C# являются настоящими динамическими массивами. Они относятся к ссылочным типам и память им отводится динамически в "куче". Однако, не снято ограничение 0-базируемости, хотя во многих задачах гораздо удобнее работать с массивами, у которых нижняя граница не равна нулю.

В языке C++ "классических" многомерных массивов нет. Вместо них введены одномерные массивы и массивы массивов, которые являются более общей структурой данных и позволяют задать не только многомерный куб, но и изрезанную, ступенчатую структуру.

В языке C#, соблюдая преемственность, сохранены одномерные массивы и массивы массивов. В дополнение к ним в язык добавлены многомерные массивы.

Простейшей формой многомерного массива является двумерный массив. Местоположение любого элемента в двумерном массиве обозначается двумя индексами. Такой массив можно предста-

вить в виде таблицы, первый индекс которого указывает на строки данной таблицы, а второй – на ее столбцы.

В следующей строке кода объявляется двумерный массив `integer` размерами 10x15:

```
int[,] table = new int[10, 15];
```

При объявлении этого массива оба его размера разделяются запятой. В первой части этого объявления синтаксическое обозначение означает, что создается переменная типа ссылки на двумерный массив. Если же память распределяется для массива с помощью оператора `new`, то используется следующее синтаксическое обозначение:

```
int[10, 15]
```

Данное объявление создает массив размерами 10×15. Для доступа к элементу двумерного массива следует указывать оба индекса, разделив их запятой.

В следующей строке кода элементу массива `myArray` с координатами положения, т. е. индексами (3, 5) присваивается значение 10:

```
myArray [3, 5] = 10;
```

В C# допускаются массивы трех и более измерений.

Общая форма объявления многомерного массива.

```
тип[, . . . , ] имя_массива = new тип[размер1, размер2,  
. . . размерN];
```

Например, в приведенном ниже объявлении создается трехмерный целочисленный массив размерами 4x10x3.

```
int [,,] multiArray = new int[4, 10, 3];
```

А в следующем операторе элементу массива `multiArray`, положение которого определяется индексами (2, 4, 1) присваивается значение 100:

```
multiDim[2, 4, 1] = 100;
```

Ниже приведен пример программы, в которой сначала организуется трехмерный массив, содержащий матрицу значений 3x3x3, а затем вычисляется сумма значений диагональных элементов этого массива.

```
// Суммировать значения по одной из диагоналей
```

```
// матрицы 3x3x3.
```

```
using System;
```

```
class Example
```

```
{
```

```
    static void Main()
```

```
    {
```

```
        int [,,] m = new int[3, 3, 3];
```

```
        int sum = 0;
```

```
        int n = 1;
```

```
        for (int i = 0; i < 3; i++)
```

```
            for (int j = 0; j < 3; j++)
```

```

        for (int k = 0; k < 3; k++)
            m[i, j, k] = n++;
        sum = m[0, 0, 0] +
            m[1, 1, 1] + m[2, 2, 2];
        Console.WriteLine("Сумма
            значений по диагонали: "
            + sum);
    }
}

```

Результат выполнения этой программы:

Сумма значений по диагонали: 42

### Инициализация многомерных массивов

Для инициализации многомерного массива достаточно заключить в фигурные скобки список инициализаторов каждого его размера. Общая форма инициализации двумерного массива имеет следующий вид:

```

тип[, ] имя_массива =
{
    {val, val, val, ..., val},
    {val, val, val, ..., val},
    {val, val, val, ..., val}
};

```

где *val* обозначает инициализирующее значение, а каждый внутренний блок – отдельный ряд. Первое значение в каждом ряду сох-

раняется на первой позиции в массиве, второе значение – на второй позиции и т. д. Блоки инициализаторов разделяются запятыми, а после завершающей эти блоки закрывающей фигурной скобки ставится точка с запятой.

Следующие два примера иллюстрируют возможность инициализации массива непосредственным вводом с клавиатуры и считыванием информации из текстового файла.

### **Пример 1.**

```
// Заполнить двумерный массив
// непосредственно вводом с клавиатуры
using System;
namespace Example6
{
    class Example6_1
    {
        static void Main()
        {
            int i, j;
            string strValue;
            int[,] iArray = new int[3, 4];
            int[,] jArray = new int[3, 4];
            for (i = 0; i < 3; ++i)
            {
                for (j = 0; j < 4; ++j)
                {
                    // ввод и присваивание значений
```



```

        strValue = Console.ReadLine();
        iArray[i, j] =
            Convert.ToInt32(strValue);
    }
}
// вывод значений массива iArray на экран
for (i = 0; i < 3; ++i)
{
    for (j = 0; j < 4; ++j)
    {
        Console.Write(" iArray[" + i + ", "
            + j + "] = " + iArray[i, j]);
    }
    Console.WriteLine();
}
Console.WriteLine();
// вывод значений массива jArray на экран
for (i = 0; i < 3; ++i)
{
    for (j = 0; j < 4; ++j)
    {
        jArray[i, j] = iArray[i, j] * 10;
        Console.Write(" jArray[" + i + ", "
            + j + "] = " + jArray[i, j]);
    }
    Console.WriteLine();
}

```

```

        Console.WriteLine();
    }
}

```

```

C:\WINDOWS\system32\cmd.exe
11
12
13
14
21
22
23
24
31
32
33
34
iArray[0, 0] = 11 iArray[0, 1] = 12 iArray[0, 2] = 13 iArray[0, 3] = 14
iArray[1, 0] = 21 iArray[1, 1] = 22 iArray[1, 2] = 23 iArray[1, 3] = 24
iArray[2, 0] = 31 iArray[2, 1] = 32 iArray[2, 2] = 33 iArray[2, 3] = 34

jArray[0, 0] = 110 jArray[0, 1] = 120 jArray[0, 2] = 130 jArray[0, 3] = 140
jArray[1, 0] = 210 jArray[1, 1] = 220 jArray[1, 2] = 230 jArray[1, 3] = 240
jArray[2, 0] = 310 jArray[2, 1] = 320 jArray[2, 2] = 330 jArray[2, 3] = 340

Для продолжения нажмите любую клавишу . . .

```

Основным методом, используемым для чтения данных с консоли, является метод `ReadLine`. Он читает с консоли строку текста, завершаемую знаком конца строки. Эта строка и является результатом, возвращаемым методом `ReadLine`. Введенная строка с помощью метода `ToInt32` конвертируется в значение типа `int` и это значение присваивается элементу `iArray[i, j]`. Далее в двойном цикле заполняется массив `jArray`

```
jArray[i, j] = iArray[i, j] * 10;
```

и затем оба массива выводятся на экран.

Для ввода данных из файла используется конструктор класса `StreamReader`. В классе `StreamReader` определено несколько конструкторов. Наиболее часто используемый `StreamReader(string имя_файла)`, где *имя\_файла* – это имя открываемого файла, включая полный путь к нему. Здесь необходимо отметить, что в имени файла присутствует служебный символ `\`, который обычно на экран не выводится. Если все же необходимо использовать служебные символы в выводимом тексте, следует перед таким символом поставить дополнительный символ `\\` (обратный слеш).

### **Пример 2.**

```
// Двухмерный массив
// Ввод массива в файл и
// вывод массива из файла
using System;
using System.IO;
namespace Example6
{
    class Example6_2
    {
        static void Main()
        {
            int i, j;
            string strValue;
            int[,] iArray1 = new int[3, 4];
            int[,] iArray2 = new int[3, 4];
```

```

StreamReader sRead = new
StreamReader("C:\\C#\\dat.txt");
StreamWriter sWrite = new
StreamWriter("C:\\C#\\res.txt");
for (i = 0; i < 3; i++)
{
    for (j = 0; j < 4; j++)
    {
        strValue = sRead.ReadLine();
        iArray1[i, j] =
        Convert.ToInt32(strValue);
        iArray2[i, j] = iArray1[i, j] *
        100;
        strValue = string.Format("\n {0,
        4:D} {1, 4:D} {2, 6:D} {3, 6:D}",
        i, j, iArray1[i, j], iArray2[i,
        j]);
        Console.WriteLine(strValue);
        Console.WriteLine();
    }
}
for (i = 0; i < 3; i++)
{
    for (j = 0; j < 4; j++)
    {
        sWrite.WriteLine(iArray2[i, j]);
    }
}

```

```

        }
        sWri te. Cl ose();
    }
}
}

```

Смысл этого примера достаточно прозрачен и отличается только тем, что вместо класса `Console` (`Example6_1`) используются классы `StreamReader` и `StreamWriter` (`Example6_2`).

Следующая программа содержит пример ввода и вывода массива из файла. Массив `iArray1` содержит три столбца, первый из которых заполнен случайными числами типа `int`, второй – случайными числами типа `double` и третий значениями функции  $\sin(x)$ . Далее формируется строковая переменная `strValue`, которая форматируется и используется для вывода массива `iArray1` на экран. Следующий оператор

```
iArray2[i , j] = iArray1[i , j] * 100;
```

инициализирует массив `iArray2` и производит запись символического потока в текстовый файл `res3.txt`.

Класс `Array` – это класс, обслуживающий массивы в пространстве имен `System`. Его свойства и методы можно использовать для массивов любого из встроенных типов.

Свойство `Length` доступно только для чтения, имеет тип `int` и содержит количество элементов массива.

```
C:\WINDOWS\system32\cmd.exe

0  0  11  1100
0  1  12  1200
0  2  13  1300
0  3  14  1400
1  0  21  2100
1  1  22  2200
1  2  23  2300
1  3  24  2400
2  0  31  3100
2  1  32  3200
2  2  33  3300
2  3  34  3400
Для продолжения нажмите любую клавишу . . .
```

Метод `Copy (Array source, Array dest, count)` копирует число элементов, задаваемых параметром *count*, из исходного массива *source* в целевой массив *dest*, начиная с первого элемента массива. Далее происходит вывод массива `iArray1` на экран.

### Пример 3.

```
// Заполнить двумерный массив
// и организовать вывод на экран.
// Ввод массива в файл и
// вывод массива из файла
```

```

using System;
using System.IO;
namespace Example6
{
    class Example6_3
    {
        static void Main()
        {
            int i, j, num;
            double x = 0;
            string strValue;
            double[,] iArray1 = new double[10, 3];
            double[,] iArray2 = new double[10, 3];
            Random rnd = new Random();
            StreamWriter sWrite = new
            StreamWriter("C:\\C#\\res3.txt");
            Console.WriteLine("\n -----
            -----");
            for (i = 0; i < 10; i++)
            {
                num = rnd.Next(1, 51);
                iArray1[i, 0] =
                Convert.ToDouble(num);
                num = (rnd.Next(1, 101) - 50) / 10;
                iArray1[i, 1] =
                Convert.ToDouble(num);
                iArray1[i, 2] = Math.Sin(x);
            }
        }
    }
}

```

```

        x = x + 0.314159;
        strValue = string.Format("\n {0, 4:D}
        {1, 10:F0} {2, 10:F2} {3, 10:F2}", i,
        iArray1[i, 0], iArray1[i, 1],
        iArray1[i, 2]);
        Console.WriteLine(strValue);
    }
    Console.WriteLine("\n -----
    -----");
    for (i = 0; i < 10; i++)
    {
        for (j = 0; j < 3; j++)
        {
            iArray2[i, j] = iArray1[i, j] *
            100;
            sWrite.WriteLine(iArray2[i, j]);
        }
    }
    num = iArray2.Length;
    Array.Copy(iArray2, iArray1, num);
    for (i = 0; i < 10; i++)
    {
        strValue = string.Format("\n {0, 4:D}
        {1, 10:F0} {2, 10:F2} {3, 10:F2}", i,
        iArray2[i, 0], iArray2[i, 1],
        iArray2[i, 2]);
        Console.WriteLine(strValue);
    }
}

```



```

    }
    Console.WriteLine("\n -----
    -----");
    sWrite.Close();
}
}
}

```

The screenshot shows a Windows command prompt window with the title bar "C:\WINDOWS\system32\cmd.exe". The window displays two tables of data, each preceded by a dashed line separator. The first table has 10 rows of data, and the second table also has 10 rows. At the bottom of the window, there is a prompt "Для продолжения нажмите любую клавишу . . . \_".

Index	Value 1	Value 2	Value 3
0	32	-3,00	0,00
1	12	-3,00	0,31
2	29	0,00	0,59
3	19	-2,00	0,81
4	46	4,00	0,95
5	1	-3,00	1,00
6	39	1,00	0,95
7	22	4,00	0,81
8	39	2,00	0,59
9	20	-4,00	0,31

Index	Value 1	Value 2	Value 3
0	3200	-300,00	0,00
1	1200	-300,00	30,90
2	2900	0,00	58,78
3	1900	-200,00	80,90
4	4600	400,00	95,11
5	100	-300,00	100,00
6	3900	100,00	95,11
7	2200	400,00	80,90
8	3900	200,00	58,78
9	2000	-400,00	30,90

Для продолжения нажмите любую клавишу . . . \_

Пример Example6\_4 демонстрирует метод вычисления суммы и среднеарифметического всех элементов массива, который заполняется случайными числами. Эти вычисления аналогичны тем, которые проводились для одномерного массива. Разница лишь в том, что в случае одномерного массива использовался одинарный цикл, а для двумерного массива – двойной.

#### Пример 4.

```
// Заполнить двухмерный массив случайными числами.
// Вычислить сумму и среднеарифметическое всех
// элементов
using System;
using System.IO;
namespace Example6
{
    class Example6_4
    {
        static void Main()
        {
            int i, j;
            int num, sum = 0;
            int[,] iArray = new int[5, 6];
            int[] iA = new int[6];
            Random rnd = new Random();
            string strValue = "\n -----
            -----";
            Console.WriteLine("\n ");
            // номера столбцов
```

```

for (j = 0; j < 6; j++)
    Console.WriteLine("{0, 5}", j);
Console.WriteLine(strValue);
// заполнение массива; сумма и
// среднеарифметическое всех элементов
for (i = 0; i < 5; i++)
{
    for (j = 0; j < 6; j++)
    {
        num = rnd.Next(1, 101);
        iArray[i, j] =
            Convert.ToInt32(num);
        iA[j] = iArray[i, j];
        sum = sum + iArray[i, j];
    }
}
// вывод массива
for (i = 0; i < 5; i++)
{
    // номер строки
    Console.WriteLine("\n " + i);
    for (j = 0; j < 6; j++)
    {
        // элементы строки
        Console.WriteLine("{0, 5}",
            iArray[i, j]);
    }
}

```

```

    }
    Console.WriteLine(strValue);
    Console.WriteLine();
    Console.WriteLine(" Сумма элементов
массива: " + sum);
    Console.WriteLine(" Среднеарифметическое:
" + sum / 30);
    Console.WriteLine();
}
}
}

```

```

C:\WINDOWS\system32\cmd.exe

0  1  2  3  4  5
-----
0  36  9  26  98  41  82
1  95  35  24  59  44  85
2  55  17  38  35  74  77
3  61  34  97  76  91  45
4  94  53  45  34  91  74
-----

Сумма элементов массива: 1725
Среднеарифметическое: 57

Для продолжения нажмите любую клавишу . . .

```

Следующий пример демонстрирует перестановку двух произвольных строк в двумерном массиве А, который инициализируется случайными числами. Все происходит аналогично перестановке любых двух элементов одномерного массива. Разница, как и раньше, лишь в том, что в случае одномерного массива использовался одинарный цикл, а для двумерного массива – двойной.

### Пример 5.

```
// Заполнить двухмерный массив A(M, N)
// случайными числами.
// Переставить в этом массиве строки 2 и M - 2
using System;
namespace Example6
{
    class Example6_5
    {
        static void Main()
        {
            int M = 6, N = 8;
            int i, j, temp;
            int[,] iArray1 = new int[M, N];
            int[,] iArray2 = new int[M, N];
            int[] iArray3 = new int[N];
            Random rnd = new Random();
            string strValue = "\n -----
            -----";
            Console.WriteLine("\n ");
            Console.WriteLine("\n Исходный массив:
            ");
            Console.WriteLine(strValue);
            // заполнение массива
            for (i = 0; i < M; i++)
            {
                for (j = 0; j < N; j++)
```

```

    {
        temp = rnd.Next(1, 101);
        iArray1[i, j] =
        Convert.ToInt32(temp);
        iArray2[i, j] =
        Convert.ToInt32(temp);
        iArray3[j] = iArray1[i, j];
    }
    foreach (int jj in iArray3)
    {
        Console.WriteLine("{0, 5}", jj);
    }
    Console.WriteLine("\n");
}
Console.WriteLine(strValue);
Console.WriteLine("\n Массив после
перестановки: ");
Console.WriteLine(strValue);
for (j = 0; j < N; j++)
{
    temp = iArray2[2, j];
    iArray2[2, j] = iArray2[M - 2, j];
    iArray2[M - 1, j] = temp;
}
for (i = 0; i < M; i++)
{
    for (j = 0; j < N; j++)

```

```

        {
            iArray3[j] = iArray2[i, j];
        }
        foreach (int jj in iArray3)
        {
            Console.WriteLine("{0, 5}", jj);
        }
        Console.WriteLine("\n");
    }
    Console.WriteLine(strValue);
    Console.WriteLine();
    Console.WriteLine();
}
}
}

```

```

C:\WINDOWS\system32\cmd.exe

Исходный массив:
-----
48  77  81  21  66  71  57  34
87  39  92  26  95  34  44   4
60  52  64  16  37  30  14  17
66  44  40  79  77  94  12  98
60  25  74  10  15  24  92   3
62  58  13  40  83  50  60  93
-----

Массив после перестановки:
-----
48  77  81  21  66  71  57  34
87  39  92  26  95  34  44   4
60  25  74  10  15  24  92   3
66  44  40  79  77  94  12  98
60  25  74  10  15  24  92   3
60  52  64  16  37  30  14  17
-----

Для продолжения нажмите любую клавишу . . .

```

### Пример 6.

```
// Заполнить двухмерный массив A(5, 6)
// случайными числами.
// Найти в этом массиве строку,
// сумма элементов которой
// является максимальной, а в ней
// минимальный по величине элемент
using System;
namespace Example6
{
    class Example6_6
    {
        static void Main()
        {
            int im = 0, jm = 0, i, j;
            int num, sum = 0;
            int max = -100, min = 100;
            int[,] iArray = new int[5, 6];
            int[] iA = new int[6];
            Random rnd = new Random();
            string strValue = "\n -----
            -----";
            Console.WriteLine("\n ");
            // номера столбцов
            for (j = 0; j < 6; j++)
                Console.WriteLine("{0, 5}", j);
            Console.WriteLine(strValue);
        }
    }
}
```



```

// заполнение массива; сумма строк
for (i = 0; i < 5; i++)
{
    sum = 0;
    for (j = 0; j < 6; j++)
    {
        num = rnd.Next(1, 101);
        iArray[i, j] =
            Convert.ToInt32(num);
        iA[j] = iArray[i, j];
        sum = sum + iArray[i, j];
    }
    if (sum > max)
    {
        max = sum;
        im = i;
    }
}
for (j = 0; j < 5; j++)
{
    if (iArray[im, j] < min)
    {
        min = iArray[im, j];
        jm = j;
    }
}
// вывод массива

```

```

for (i = 0; i < 5; i++)
{
    // номер строки
    Console.WriteLine("\n " + i);
    for (j = 0; j < 6; j++)
    {
        // элементы строки
        Console.WriteLine("{0, 5}",
            iArray[i, j]);
    }
}
Console.WriteLine(strValue);
Console.WriteLine();
Console.WriteLine("\n i = " + i m);
Console.WriteLine("\n j = " + j m);
Console.WriteLine();
}
}
}

```

Сумма элементов строки вычисляется во внутреннем цикле по  $j$ . На каждом шаге цикла к сумме элементов строки  $i$  добавляется очередной элемент. По окончании цикла переменная  $sum$  будет содержать полную сумму всех элементов данной строки. Максимальная сумма определяется во внешнем цикле. Для этого используется следующая группа операторов

```

if (sum > max)

```

```
{
    max = sum;
    i m = i ;
}
```

которая определяет максимальную сумму и номер строки  $i m$ . Другая группа операторов

```
i f (i Array[i m, j ] < mi n)
{
    mi n = i Array[i m, j ];
    j m = j ;
}
```

используется для поиска минимального элемента и его номера  $j m$  в найденной строке.

The screenshot shows a Windows command prompt window with the title bar "C:\WINDOWS\system32\cmd.exe". Inside the window, a 6x6 array of numbers is displayed, with indices 0 to 5 for both rows and columns. The array is as follows:

	0	1	2	3	4	5
0	69	43	1	51	26	5
1	24	26	49	74	9	3
2	39	92	47	32	74	29
3	59	13	79	43	23	18
4	7	50	71	10	73	25

Below the array, the current values of  $i$  and  $j$  are shown:

```
i = 2
j = 3
```

At the bottom, there is a prompt: "Для продолжения нажмите любую клавишу . . ."

Генерация таблицы умножения достаточно тривиальная задача. Каждый элемент таблицы определяется произведением номера

строки на номер столбца  $k = i * j$ . Единственная трудность заключается в организации вывода таблицы на экран.

Конструкция выбора

```
if (k < 10) Console.WriteLine("  ");  
else      Console.WriteLine(" ");
```

вставляет перед числом  $k < 10$  два пробела, а во всех остальных случаях – один пробел. На нижеследующем рисунке представлен результат вывода, полученный при выполнении данной программы.

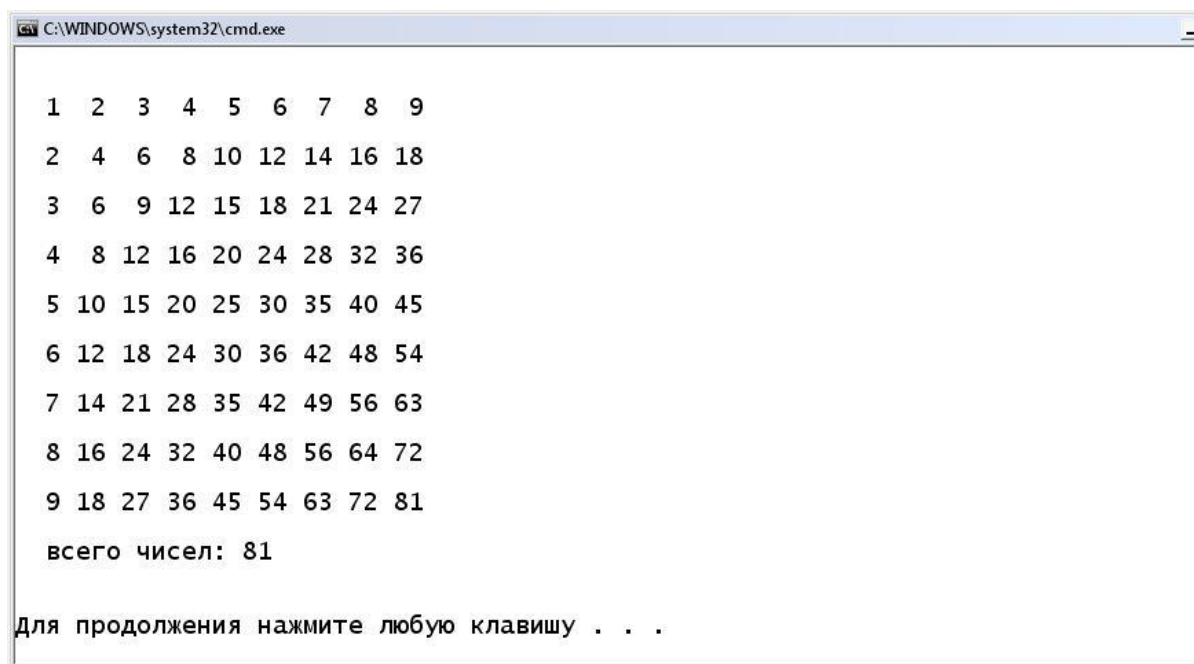
### Пример 7.

```
// Рассчитать и вывести на экран  
// таблицу умножения. Подсчитать общее  
// количество чисел в данной таблице  
using System;  
namespace Example6  
{  
    class Example6_7  
    {  
        static void Main()  
        {  
            int i, j, k, m = 0;  
            for (i = 1; i < 10; i++)  
            {  
                Console.WriteLine("\n");  
                for (j = 1; j < 10; j++)  
                {  
                    k = i * j;
```

```

        m++;
        if (k < 10) Console.WriteLine(" ");
        else Console.WriteLine(" ");
        Console.WriteLine(k);
    }
}
Console.WriteLine("\n\n  всего чисел: "
+ m);
Console.WriteLine("\n");
}
}
}

```



```

C:\WINDOWS\system32\cmd.exe

1  2  3  4  5  6  7  8  9
2  4  6  8 10 12 14 16 18
3  6  9 12 15 18 21 24 27
4  8 12 16 20 24 28 32 36
5 10 15 20 25 30 35 40 45
6 12 18 24 30 36 42 48 54
7 14 21 28 35 42 49 56 63
8 16 24 32 40 48 56 64 72
9 18 27 36 45 54 63 72 81
всего чисел: 81
Для продолжения нажмите любую клавишу . . .

```

Программа данного примера выводит на экран верхнюю и нижнюю треугольные матрицы. Для управления выводом используются операторы `break` и `continue`. Оператор `break` прерывает

выполнение цикла при выполнении заданного условия. Оператор `continue` также при выполнении заданного условия приводит к пропуску следующих за ним операторов блока, но без выхода из блока.

### **Пример 8.**

```
// Рассчитать таблицу (матрицу) умножения.
// Вывести на экран верхнюю треугольную и нижнюю
// треугольную матрицы. Подсчитать общее
// количество чисел (элементов) в каждой из матриц.
using System;
namespace Example6
{
    class Example6_8
    {
        static void Main()
        {
            // верхняя треугольная матрица
            int i, j, k, m = 0;
            for (i = 1; i < 10; i++)
            {
                Console.WriteLine("\n");
                for (j = 1; j < 10; j++)
                {
                    k = i * j;
                    if (j > 10 - i) break;
                    m++;
                    if (k < 10) Console.Write(" ");
                }
            }
        }
    }
}
```

```

        else Console.WriteLine(" ");
        Console.WriteLine(k);
    }
}
Console.WriteLine("\n\n  всего чисел: "
+ m);
Console.WriteLine("\n");
m = 0;
for (i = 1; i < 10; i++)
{
    Console.WriteLine("\n");
    for (j = 1; j < 10; j++)
    {
        k = i * j;
        if (k < i * (10 - i))
            Console.WriteLine(" ");
        else
        {
            if (k < 10) Console.WriteLine
                (" " + k);
            else Console.WriteLine(" " + k);
            m++;
        }
    }
}
Console.WriteLine("\n\n  всего чисел: "
+ m);

```

```

    Console.WriteLine("\n");
}
}
}

```

```

C:\WINDOWS\system32\cmd.exe

1 2 3 4 5 6 7 8 9
2 4 6 8 10 12 14 16
3 6 9 12 15 18 21
4 8 12 16 20 24
5 10 15 20 25
6 12 18 24
7 14 21
8 16
9
всего чисел: 45

          9
        16 18
      21 24 27
    24 28 32 36
  25 30 35 40 45
24 30 36 42 48 54
21 28 35 42 49 56 63
16 24 32 40 48 56 64 72
9 18 27 36 45 54 63 72 81
всего чисел: 45

Для продолжения нажмите любую клавишу . . .

```

В приведенных выше примерах применения двумерного массива, по существу создавался так называемый прямоугольный массив. Двумерный массив можно представить в виде таблицы, в кото-



рой длина каждой строки остается неизменной по всему массиву. Но в С# можно также создавать специальный тип двумерного массива, называемый ступенчатым массивом. Ступенчатый массив представляет собой массив массивов, в котором длина каждого массива может быть разной. Следовательно, ступенчатый массив можно использовать для составления таблицы из строк разной длины.

Ступенчатые массивы объявляются с помощью ряда квадратных скобок, в которых указывается их размерность. Например, для объявления двумерного ступенчатого массива служит следующая общая форма:

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

где *размер* обозначает число строк в массиве. Память для самих строк распределяется индивидуально, и поэтому длина строк может быть разной. Память сначала распределяется для его первого измерения автоматически, а затем для второго измерения вручную.

После создания ступенчатого массива доступ к его элементам осуществляется по индексу, указываемому в отдельных квадратных скобках. Например, в следующей строке кода элементу массива *jagged*, находящемуся на позиции с координатами (2,1), присваивается значение 10:

```
jagged[2][1] = 10;
```

Обратите внимание на синтаксические отличия в доступе к элементу ступенчатого и прямоугольного массива.

В приведенном ниже примере программы демонстрируется создание двумерного ступенчатого массива. Здесь следует обратить внимание, что язык C# позволяет использовать в качестве имен переменных и массивов русские имена.

### **Пример 9.**

```
// Инициализировать и вывести на экран
// ступенчатый массив.
using System;
namespace Example6
{
    class Example6_9
    {
        static void Main()
        {
            int j, max = 0;
            int[] Массив = new int[5];
            for (j = 0; j < 5; j++)
            {
                Массив[j] = j + 2;
                if (Массив[j] > max) max = Массив[j];
            }
            // вспомогательный массив, определяющий
            // границы каждой из строк ступенчатого
            // массива
            int размер0 = 5;
            int размер1 = max;
            int размер2 = Convert.ToInt32((4 * max
```

```

- 1) / 3);
int[] размер = { размер0, размер1,
размер2 };
int[ ][ ] стМассив = new int[3][ ];
стМассив[0] = new int[размер[0]];
стМассив[1] = new int[размер[1]];
стМассив[2] = new int[размер[2]];
Console.WriteLine();
Console.WriteLine(" ");
for (j = 0; j < размер2; j++)
Console.WriteLine("{0,5}", j);
Console.WriteLine("\n\t\t -----
----- ");
Console.WriteLine(" стМассив[0][5]");
for (j = 0; j < размер0; j++)
стМассив[0][j] = j + 1;
foreach (int jj in стМассив[0])
Console.WriteLine("{0,5}", jj);
Console.WriteLine();
Console.WriteLine(" стМассив[1][6]");
for (j = 0; j < размер1; j++)
стМассив[1][j] = j + 11;
foreach (int jj in стМассив[1])
Console.WriteLine("{0,5}", jj);
Console.WriteLine();
Console.WriteLine(" стМассив[2][7]");
for (j = 0; j < размер2; j++)

```

```

        стМассив[2][j] = j + 111;
        foreach (int jj in стМассив[2])
        Console.WriteLine("{0,5}", jj);
        Console.WriteLine("\n\t\t -----
        ----- ");
        Console.WriteLine();
    }
}
}

```

```

C:\WINDOWS\system32\cmd.exe

        0    1    2    3    4    5    6
-----
стМассив[0][5]  1    2    3    4    5
стМассив[1][6] 11   12   13   14   15   16
стМассив[2][7] 111  112  113  114  115  116  117
-----
Для продолжения нажмите любую клавишу . . .

```

Ступенчатые массивы находят полезное применение не во всех, а лишь в некоторых случаях. Так, если требуется очень длинный двумерный массив, который заполняется не полностью, т.е. такой массив, в котором используются не все, а лишь отдельные его элементы, то для этой цели идеально подходит ступенчатый массив.

Следует отметить, что ступенчатые массивы представляют собой массивы массивов, и поэтому они не обязательно должны состоять из одномерных массивов. Так в приведенной ниже строке кода создается массив двумерных массивов.

```
int[ ][, ] jArray = new int[5][,];
```

## 2. Практическая часть

### Задания к лабораторной работе

Для заданных условия составить процедуру и придумать несколько наборов тестовых данных для отладки. Возможно использование как статических массивов, так и динамических. Ввод исходных данных осуществить из файла или с клавиатуры.

1. Написать процедуру вычисления суммы всех элементов для заданного двумерного массива  $A$  состоящего из 5-ти строк и 6-ти столбцов.
2. Написать процедуру вычисления для заданного массива  $A(5, 6)$  среднего арифметического элементов значения его положительных элементов. Известно, что хотя бы один элемент массива имеет положительное значение
3. Для заданного массива  $A(4, 6)$  вычислить среднее арифметическое значение положительных элементов каждой строки. Результаты поместить в одномерный массив  $B(4)$ . Известно, что в каждой строке массива хотя бы один элемент имеет положительное значение.
4. Вычислить для заданного массива  $B(20, 30)$  наибольший элемент каждого столбца. Результаты поместить в одномерный массив  $BM(30)$ .
5. Написать процедуру поиска для заданного массива  $A(4, 5)$ , строки с наибольшим средним арифметическим значением положительных элементов. Результат поместить в одномерный массив  $D(5)$ . Считаем, что хотя бы один положительный элемент в каждой строке имеется.

6. Написать процедуру. Для двумерного массива  $A(6, 4)$ , переставить местами 2-ю и 4-ю строки.
7. Написать процедуру. Для заданного массива  $B(4, 5)$ , переставить местами столбец с наибольшим количеством нулевых элементов и столбец последний по порядку следования в массиве  $B$ .
8. Написать процедуру для подсчета в заданном массиве  $A(5, 5)$  количества элементов превышающих заданное число  $B$  и лежащих на главной диагонали и выше нее.
9. Написать процедуру для вывода на печать отрицательных элементов, лежащих на главной диагонали заданного массива  $A(4, 4)$ .
10. Написать процедуру перестановки элементов заданного массива  $B(4, 4)$ , лежащих на главной и побочной главной диагонали.
11. Написать процедуру пересылки двумерного массива  $A(5, 6)$  в одномерный массив  $B(30)$  того же размера, по строкам с сохранением порядка следования элементов.
12. Написать процедуру по транспонированию заданной квадратной матрицы  $A$  максимальная размерность которой 100.
13. Написать процедуру перемножения матрицы  $A$  на вектор  $B$ .
14. Написать процедуру перемножения матрицы  $A(5, 10)$  на матрицу  $B(10, 4)$ .
15. Задан массив размером  $N \times N$ . Разделить элементы каждого столбца на элемент главной диагонали расположенный в соответствующем столбце.

16. Задан двумерный массив размером  $N \times N$ . Сформировать из его диагональных элементов одномерный массив.
17. Задан массив размером  $N \times N$ . Определить максимальный и минимальный элементы главной диагонали и переставить местами столбцы в которых лежат эти элементы.
18. Просуммировать элементы заданного двумерного массива размером  $N \times N$ , расположенные в его верхней части, ограниченной главной и побочной диагоналями, включая элементы, расположенные на этих диагоналях.
19. Для заданного двумерного массива размером  $N \times N$  просуммировать элементы, расположенные на диагоналях, параллельных главной. Результаты поместить в одномерный массив.
20. В заданном двумерном массиве размером  $N \times M$  поменять местами элементы первого и второго столбца, третьего и четвертого и т. д.
21. Задан массив размером 16. Сформировать из него массив размером  $4 \times 4$  по строкам.
22. В заданном двумерном массиве размером  $N \times M$  переместить нулевые элементы каждого столбца в конец столбца.
23. Задан двумерный массив размером  $N \times N$ . Максимальный элемент каждой строки поменять местами с диагональным элементом соответствующей строки.
24. Поместить в одномерный массив элементы, расположенные по периметру заданного двумерного массива размером  $N \times N$ . Использовать один цикл.
25. Заданный двумерный массив размером  $N \times N$  повернуть вправо на  $90^\circ$ , без использования вспомогательных массивов.

26. В заданном двумерном массиве размером  $N \times N$  поменять местами элементы, расположенные в верхней и нижней частях массива, между главной и побочной диагоналями за исключением элементов, расположенных на этих диагоналях.
27. Двумерный массив размером  $N \times N$  задан в виде одномерного массива по столбцам. Вывести на печать верхний треугольник массива по строкам, включая элементы главной диагонали.
28. Написать процедуру, задающую треугольную матрицу произвольного размера. Число, определяющее размер матрицы вводится с клавиатуры во время выполнения процедуры
29. Сформировать двумерный массив, у которого элементы равны произведению своих индексов
30. Найти сумму элементов главной и произведение элементов побочной диагоналей квадратной матрицы



## ЛАБОРАТОРНАЯ РАБОТА № 7

### Классы и методы

#### 1. Краткие теоретические сведения

Класс является обобщенным понятием, определяющим характеристики и поведение некоторого множества конкретных объектов этого класса, называемых экземплярами, или объектами, класса.

В целях ускорения работы программиста и упрощения применяемых алгоритмов применяют классы. Применение классов позволяет разбить сложную задачу на ряд простых, взаимосвязанных задач. Прodelав эту операцию неоднократно применительно к вновь полученным классам, можно получить алгоритм решения задачи в виде набора простых, понятных, легко осуществимых классов, взаимодействующих друг с другом.

«Классический» класс содержит данные, задающие свойства объектов класса, и функции, определяющие их поведение. В последнее время в класс часто добавляется третья составляющая – события, на которые может реагировать объект класса. Это оправдано для классов, использующихся в программах, построенных на основе событийно-управляемой модели, например, при программировании для Windows.

Процедура объявления и создания экземпляров пользовательского типа не отличается от таковой для предопределенных типов за исключением необходимости использования Спецификатора New (new) для создания экземпляров пользовательских типов значений и ссылочных типов.

## Спецификаторы классов:

1. **new** – используется для вложенных классов. Задаёт новое описание класса взамен унаследованного от предка. Применяется в иерархиях объектов.

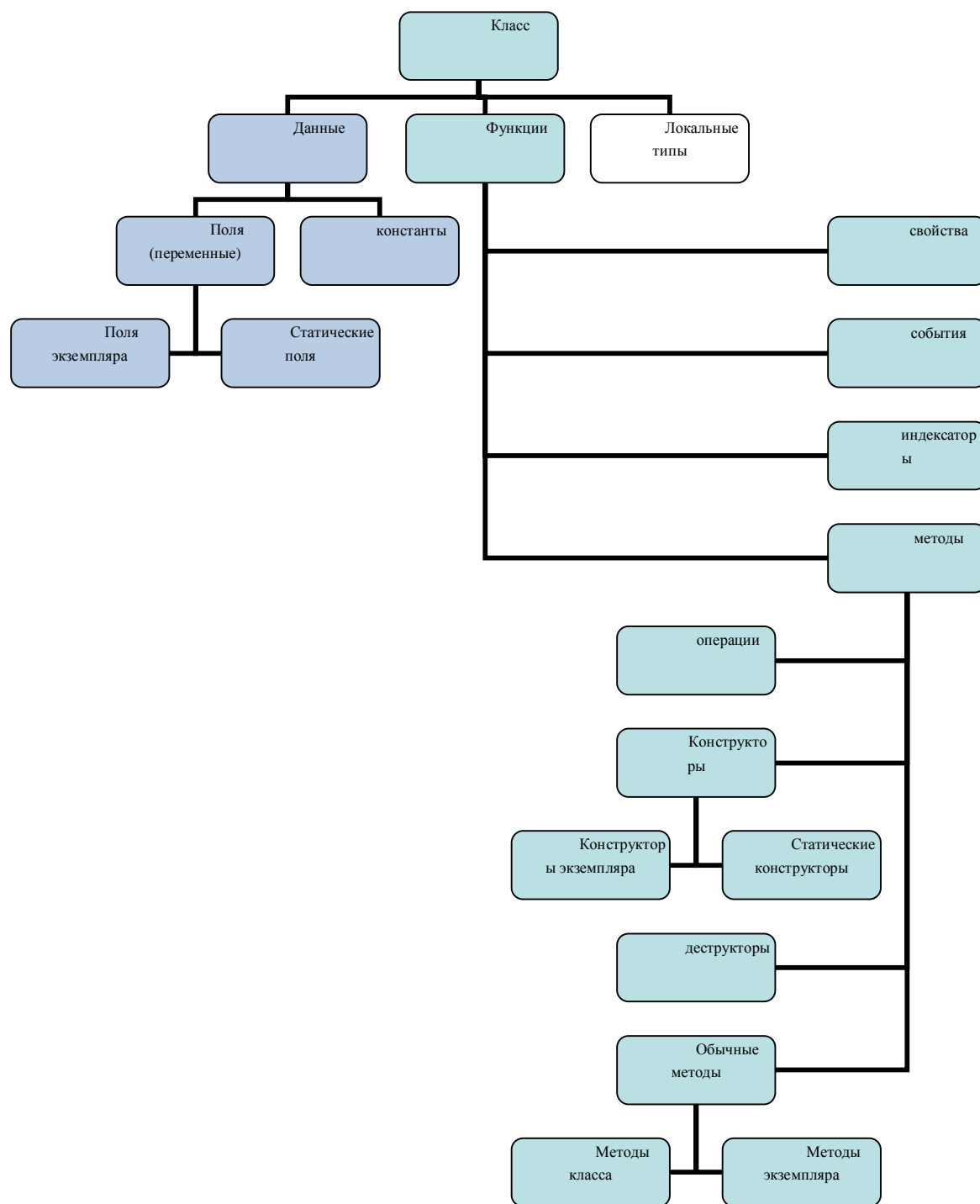


Рис. 7.1. Класс

2. **public** – доступ не ограничен
3. **protected** – используется для вложенных классов. Доступ только из элементов данного и производных классов
4. **internal** – доступ только из данной программы (сборки)
5. **protected internal** – доступ только из данного и производных классов или из данной программы (сборки)
6. **private** – используется для вложенных классов. Доступ только из элементов класса, внутри которого описан данный класс
7. **abstract** – абстрактный класс. Применяется в иерархиях объектов.
8. **sealed** – бесплодный класс. Применяется в иерархиях объектов.
9. **static** – статический класс. Введен в версию языка 2.0.

Спецификаторы 2 – 6 называются спецификаторами доступа. Они определяют, откуда можно непосредственно обращаться к данному классу. Спецификаторы доступа могут присутствовать в описании только в вариантах, приведенных в таблице, а также могут комбинироваться с остальными спецификаторами.

## Методы

Методы делают всю работу, для исполнения которой предназначены классы и структуры: вычисляют значения, изменяют данные, получают вводимую информацию – в общем, выполняют все действия, составляющие поведение объекта

## Объявление методов

Один из видов членов, которые добавляются к классам, – это методы, определяющие действия, которые должен выполнять

класс. Есть две разновидности методов: те, что возвращают значения, и те, которые никаких значений не возвращают. Следующий пример иллюстрирует обе разновидности методов:

```
// Этот метод возвращает целые (int) значения
```

```
public int Add (int first, int second)
```

```
{
```

```
    int Result;
```

```
    Result = first + second;
```

```
    Return result;
```

```
}
```

```
public void myVoidMethod ()
```

```
// void – не возвращает значений
```

```
{
```

```
    MessageBox.Show ("Этот метод не возвращает  
    значения ");
```

```
}
```

Метод является самостоятельной частью кода, которая имеет имя и может содержать аргументы, выполнять последовательность инструкций (операторов) и изменять значения своих аргументов.

### **Вызов методов.**

Метод не выполняется, пока его не вызовут. Чтобы вызвать метод, следует указать его имя и задать необходимые ему параметры.

```
// Так вызывают метод Rotate с двумя параметрами
```

```
Rotate (45, "Degrees");
```

Метод Main – особый, он вызывается сразу после начала исполнения программы. К особым методам также относятся деструк-

торы, которые вызывают во время выполнения непосредственно перед уничтожением объекта. Третий вид особых методов – конструкторы; они исполняются во время инициализации объектов.

Метод может иметь один или несколько параметров. Параметр – это аргумент, передаваемый методу вызывающим его методом. При объявлении метода список параметров указывают после его имени в круглых скобках, при этом для каждого параметра необходимо указать его тип.

```
public void DisplayName (string name, byte age)
{
    Console.WriteLine ("Hello " + name + ". You are "
        + age.ToString () +"years old.");
}
```

**Внимание.** У этого метода два параметра: один типа String, с локальным именем name, а второй, age, – типа Byte. Их область видимости ограничена методом, вне которого переменные не доступны.

Составим фрагмент программы вычисления следующих функций:

$$C = \frac{4,5e^{x^2}}{e^x + 8,5 \sin(x)} \quad \text{и} \quad D = \frac{1}{8e^{x^4} - \sin(x^4) + x^3}$$

Метод fnc типа double

```
public double fnc(a as double, x as double, z double)
{
    return (a*exp(x) + z * sin(x));
}
```

Использование метода

. . .

c = fnc ( a, x\*x, 0. ) / fnc ( 1., x, 8.5 )

b = -1.

d = 1/( fnc (8., x^4, b) + x ^ 3)

. . .

Метод с именем fnc и формальными параметрами a, x, z используется при вычислении c и d.

**Пример 1.** Задан вектор D(5). Создать класс vect осуществляющий ввод данных с клавиатуры, вывод элементов вектора на консоль, и перестановку первого по порядку элемента со k-тым отрицательным элементом вектора. Ограничение доступа в класс (снаружи можно вызывать только public части класса)

```
using System;
```

```
namespace ConsoleApplication1
```

```
{
```

```
class Program
```

```
{
```

```
    class vect          // класс векторов vect
```

```
    {
```

```
        double[] a; // объявление массива
```

```
        int n;      // размер массива
```

```
        public vect(int n)
```

```
        // конструктор создаёт вектор
```

```
        {
```

```
            this.a = new double[n];
```

```
            this.n = n;
```

```
        }
```

```

public int vv()
// заполнение вектора целиком,
// клавиатурный ввод
{
    for (int i = 0; i < n; i++)
    {
        string buf;
        // строка символов
        buf = Console.ReadLine();
        // заполнение строки
        a[i] = Convert.ToDouble(buf);
        // преобразование строки в число
    }
    return (0);
}
public int vyv()
// вывод на консоль
// элементов вектора
{
    for (int i = 0; i < n; i++)
        Console.WriteLine("a["+i+"]=" +
            a[i]);
    return (0);
}
public int perest(int k)
// переставить 1 и

```

```

// k-ое отриц. число
{
// ключ; 0  если порядок
//      -1,  если - нет
int kl = 0;
// k больше размера вектора
if ( k >= n ) kl = -1;
else
{
    // счетчик отрицательных чисел
    int cnt = 0;
    // указатель на позицию в массиве
    int pos = -1;
    double tmp;
    // подсчёт отрицательных чисел
    while ( cnt < k )
    {
        pos++;
        if (a[pos] < 0) cnt++;
    }
    tmp = a[pos];    // подсчёт
    a[pos] = a[0];   // перестановка
    a[0] = tmp;
    kl = 0;
}
return kl;
}

```



```

// главная программа
static void Main()
{
    // создали
    vect d = new vect(5);
    d.vv();           // заполнили
    d.vyv();          // вывели на
                      // консоль
    d.perest(2);       // переставили
    d.vyv();           // вывели на
                      // консоль
}
}
}

```

**Пример 2.** Задана матрица  $D(3, 3)$ . Написать метод транспонирования матрицы и метод перемножения квадратных матриц. Получить результат умножения транспонированной матрицы на соответствующую ей исходную. Отдельный класс для матриц не создаём. Работаем в классе `myprog`. Поэтому все содержимое класса доступно.

```

using System;
// моё пространство имён
namespace ConsoleApplication3
{
    class myprog

```

```
{
```

```
static void Main()
{
    // объявление матриц
    double[, ] d = new double[3, 3];
    double[, ] dt = new double[3, 3];
    double[, ] ddt = new double[3, 3];
    // заполнение d с клавиатуры
    int i; int j; // строки, колонки
    for (i = 0; i < 3; i++)
    {
        for (j = 0; j < 3; j++)
            d[i, j] =
                Convert.ToDouble(Console.ReadLine());
    }
    // транспонирование матрицы
    transp(d, 3, dt);
    // умножение транспонированной
    // матрицы dt на исходную d
    mt_q_mult(dt, d, ddt, 3);
}
// транпонирование
static void transp(double[, ] mtr,
int p, double[, ] mtrt)
// double[, ] mtr - так определяют матрицу в
// формальных параметрах
```

```

{
    int i; int j;
    for (i = 0; i < p; i++)
    {
        for (j = 0; j < p; j++)
            mtrt[j, i] = mtr[i, j];
    }
}

// умножение квадратных матриц
static void mt_q_mult(double[,] m_a,
double[,] m_b, double[,] m_r, int p)
// m_r=m_a * m_b , int p размер матриц
{
    int i; int j; int k; double mel;
    for (i = 0; i < p; i++) // строки m_a
    {
        // столбцы m_b
        for (j = 0; j < p; j++)
        {
            mel=0;
            // столбцы m_a и строки m_b
            for (k = 0; k < p; k++)
                mel=mel+m_a[i, k]*m_b[k, j];
            m_r[i, j]=mel;
        }
    }
}

```

```
}  
}
```

## 2. Практическая часть

### Задание к лабораторной работе

Написать программу, осуществляющую заданные вычисления с использованием процедур. Вид используемых классов и методов определить самостоятельно.

1. Заданы две матрицы  $A(4, 4)$  и  $B(4, 4)$ . Написать программу суммирования двух векторов  $X$  и  $Y$ , где  $X$  – вектор, в котором размещены элементы столбца матрицы  $A$  с минимальным средним арифметическим значением,  $Y$  – то же для матрицы  $B$ .
2. Заданы две матрицы  $A(4, 4)$  и  $B(4, 4)$ . Написать программу вычисления вектора  $Z = X + Y$ , где  $X$  – строка матрицы  $A$ , включающая минимальный элемент ее главной диагонали,  $Y$  – то же для матрицы  $B$ .
3. Заданы две матрицы  $A(4, 4)$  и  $B(3, 3)$  и два вектора  $C(4)$  и  $D(3)$ . Написать программу вычисления произведений матриц на соответствующие им вектора. Определить минимальное среднее арифметическое значение для полученных векторов.
4. Заданы две матрицы  $B(4, 4)$  и  $D(3, 3)$ . Написать программу транспонирования каждой из заданных матриц с последующим перемножением транспонированной матрицы на соответствующую ей исходную.

5. Заданы две матрицы  $A(3, 3)$  и  $B(3, 3)$ . Написать программу проверки – является ли произведение этих матриц перестановочным, т.е. проверить равенство  $AB = BA$ .
6. Заданы две матрицы  $C(4, 4)$  и  $D(3, 3)$ . Написать программу определения количества симметричных матриц. Матрица называется симметричной, если транспонированная матрица равна исходной. Для каждой симметричной матрицы вычислить сумму элементов, лежащих вне главной диагонали.
7. Заданы два массива  $C(6)$  и  $D(10)$ . Для каждого из них осуществить перестановку первого по порядку элемента со вторым отрицательным элементом массива.
8. Заданы три квадратных уравнения  $AX^2 + BX + C = 0$ ;  $DX^2 + EX + F = 0$  и  $ZX^2 + YX + S = 0$ . Написать программу нахождения максимального значения корня среди действительных корней этих уравнений.
9. Заданы два вектора  $A(0.052; 0.9; 0.15; 0.84; 0.67)$  и  $B(0.948; 0.1; 0.33; 0.16; 0.85)$ . Написать программу, позволяющую расположить элементы одного вектора по возрастанию, а другого – по убыванию. Вычислить сумму полученных векторов.
10. Заданы два двумерных массива  $A(4, 4)$  и  $B(3, 3)$ . Для каждого из них переставить столбцы с максимальным и минимальными элементами.
11. Заданы координаты четырёх точек  $A, B, C, D$  на плоскости. Определить наибольший из периметров треугольников  $ABC, ABD, ACD$ .

12. Три треугольника заданы координатами своих вершин. Написать программу вычисления площадей треугольников и определения минимальной площади.
13. Заданы два двумерных массива  $A(4, 4)$  и  $B(3, 3)$ . Для каждого из них переставить последнюю строку со строкой с максимальным средним арифметическим значением.
14. Заданы две матрицы  $A(4, 4)$  и  $B(4, 4)$ . Написать программу вычисления вектора  $Z = X - Y$ , где  $X$  – столбец матрицы  $A$ , включающий минимальный элемент матрицы;  $Y$  – то же для матрицы  $B$ .
15. Заданы три вектора  $X, Y, Z$ . Написать программу вычисления  $S = \sum_{i=1}^n C_i \cdot D_i$ , где  $C_i$  и  $D_i$  – компоненты векторов  $C$  и  $D$ , которые соответственно равны  $C = X + Y$ ;  $D = X - Z$
16. Заданы матрицы  $C(4, 4)$  и  $D(3, 3)$ . Определить индексы максимального элемента каждой из матриц среди элементов, расположенных выше главной диагонали.
17. Заданы вектора  $A(5), B(10), D(15)$ . Для каждого из них определить максимальный и минимальный элементы и их индексы.
18. Заданы координаты вершин трех треугольников. Определить сколько треугольников лежит внутри окружности радиуса  $R$  с центром в начале координат.
19. Заданы координаты десяти точек плоскости и координаты точки-полюса. Найти точку, максимально удалённую от полюса, среди первых четырёх заданных точек и такую же точку – среди последних шести. Определить также расстояние между найденными точками.

20. Заданы массивы  $Y(4, 4)$  и  $Z(8, 8)$ . Для каждого из них вычислить среднее арифметическое значение положительных элементов, расположенных выше главной диагонали.
21. Заданы координаты вершин трех треугольников. Определить треугольник с максимальным периметром.
22. Заданы три матрицы размерами  $2 \times 2$ ,  $3 \times 3$ ,  $4 \times 4$ . Для каждой из матриц определить среднее арифметическое значение положительных элементов главной диагонали.
23. Заданы три одномерных массива разной размерности. Для каждого из массивов определить повторяющиеся элементы.
24. Заданы координаты четырех точек на плоскости. Определить номера точек, расстояние между которыми минимальное.
25. Заданы две окружности  $(x - a)^2 + (y - b)^2 = r^2$  и  $(x - c)^2 + (y - d)^2 = R^2$  и координаты четырех точек. Определить количество точек, лежащих внутри каждой окружности.
26. Заданы три одномерных массива  $A(5)$ ,  $B(10)$ ,  $C(15)$ . Написать программу решения уравнения  $px^2 + dx + r = 0$ , где  $p$  – минимальный элемент матрицы  $A$ ,  $d$  – минимальный элемент матрицы  $B$ ,  $r$  – минимальный элемент матрицы  $C$ .
27. Заданы одномерные массивы  $X(5)$  и  $Y(7)$ . Для каждого из них определить количество и сумму элементов, которые без остатка делятся на заданное число  $B$ .
28. Составить программу заполнения массивов  $A(5)$  и  $B(10)$  факториалами значений индексов их элементов. Вычисление факториала выполнить в подпрограмме.

29. Заданы матрицы  $A(4, 4)$ ,  $B(6, 6)$ . Для каждой матрицы определить среднее арифметическое значение положительных элементов, расположенных не выше главной диагонали.
30. Матрицами  $F(2, 2)$  и  $E(2, 2)$  в декартовой системе координат заданы две окружности положением своего центра и точки на дуге. Вычислить параметры окружностей.



## ЛИТЕРАТУРА

1. Архипов В. Н., Калядин В. И., Любин А. Н., Макаров А. И. Программирование на Фортране: методические указания к лабораторным работам. – М.; МГТУ «МАМИ», 2007, – 144 с.
2. Калядин В. И., Макаров А. И. Основы работы на персональном компьютере: сборник лабораторных работ. – М.; МГТУ «МАМИ», 2010, – 85 с.
3. Лобанов А. С., Туманова М. Б. Решение задач на языке Visual Basic for Application: учебное пособие. – М.; МГТУ «МАМИ», 2009, – 90 с.
4. Павловская Т. А. С# Программирование на языке высокого уровня. ПИТЕР, 2007, – 432 с.
5. Герберт Шмидт. Полное руководство С# 3.0. М.; Изд. дом «Вильямс», 2010, – 992.

## ОГЛАВЛЕНИЕ

ВВЕДЕНИЕ .....	3
ПРАВИЛА ВЫПОЛНЕНИЯ РАБОТ .....	20
ЛАБОРАТОРНАЯ РАБОТА № 1.....	22
ЛАБОРАТОРНАЯ РАБОТА № 2.....	41
ЛАБОРАТОРНАЯ РАБОТА № 3.....	58
ЛАБОРАТОРНАЯ РАБОТА № 4.....	87
ЛАБОРАТОРНАЯ РАБОТА № 5.....	105
ЛАБОРАТОРНАЯ РАБОТА № 6.....	140
ЛАБОРАТОРНАЯ РАБОТА № 7.....	177
ЛИТЕРАТУРА .....	193

*Учебное издание*

Антони Валерий Иосифович  
Архипов Владимир Николаевич  
Любин Александр Николаевич  
Тихомиров Василий Николаевич

**ОСНОВЫ ПРОГРАММИРОВАНИЯ НА C#**  
**Сборник лабораторных работ**

*Под редакцией авторов*

*Оригинал-макет подготовлен редакционно-издательским отделом МГТУ  
«МАМИ»*

По тематическому плану внутривузовских изданий учебной литературы на 2011 г.

Подписано в печать 27.09.11. Формат 60х90 1/16. Бумага 80г/м<sup>2</sup>

Гарнитура «Таймс». Ризография. Усл. печ. л. 12.1

Тираж 100 экз. Заказ №.119-11.

МГТУ «МАМИ»

107023, г. Москва, Б. Семеновская ул., 38