Е. В. Кудрина, М. В. Огнева

ОСНОВЫ АЛГОРИТМИЗАЦИИ
И ПРОГРАММИРОВАНИЯ НА ЯЗЫКЕ C#

УЧЕБНОЕ ПОСОБИЕ
ДЛЯ БАКАЛАВРИАТА И СПЕЦИАЛИТЕТА

Рекомендовано Учебно-методическим отделом высшего образования в качестве учебного пособия для студентов высших учебных заведений, обучающихся по ИТ-направлениям

ВЫБОР РЕДАКЦИИ

Москва ▪ Юрайт ▪ 2019

УДК 004.421(075.8)
ББК 32.973я73
К88

***Авторы*:**

Кудрина Елена Вячеславовна — доцент кафедры информатики и программирования факультета компьютерных наук и информационных технологий ФГБОУ ВО «СГУ имени Н. Г. Чернышевского» (г.Саратов);

Огнева Марина Валентиновна — кандидат физико-математических наук, заведующая кафедрой информатики и программирования факультета компьютерных наук и информационных технологий ФГБОУ ВО «СГУ имени Н. Г. Чернышевского» (г.Саратов).

***Рецензенты*:**

Андрейченко Д. К. — доктор физико-математических наук, профессор, заведующий кафедрой математического обеспечения вычислительных комплексов и информационных систем на базе филиала ООО «Эпам Систэмз» в г. Саратове ФГБОУ ВО «СГУ имени Н. Г. Чернышевского»;

Кондратов Д. В. — доктор физико-математических наук, доцент, заведующий кафедрой прикладной информатики в управлении Поволжского института управления имени П. А. Столыпина — филиала Российской академии народного хозяйства и государственной службы при Президенте Российской Федерации.

Кудрина, Е. В.

К88 Основы алгоритмизации и программирования на языке C# : учеб. пособие для бакалавриата и специалитета / Е. В. Кудрина, М. В. Огнева. — М. : Издательство Юрайт, 2019. — 322 с. — (Серия : Бакалавр. Академический курс).

ISBN 978-5-534-09796-2

Учебное пособие представляет собой учебно-методическую разработку, которая, с одной стороны, направлена на изложение основ программирования на языке C#, а с другой стороны — на формирование навыков применения базовых алгоритмов для решения практико-ориентированных задач. Для освоения материала данного учебника не нужны никакие предварительные знания по программированию. Простота изложения материала и большое количество разобранных примеров делают изучение языка C# доступным для широкого круга читателей.

Содержание учебного пособия соответствует актуальным требованиям Федеральных государственных образовательных стандартов высшего образования для IT-направлений.

Для студентов высших учебных заведений, обучающихся по ИТ-направлениям.

УДК 004.421(075.8)
ББК 32.973я73

ISBN 978-5-534-09796-2

Оглавление

Предисловие

При решении прикладных задач из разных областей, при создании программного обеспечения вопрос о выборе подходящего алгоритма и о том, с помощью какой структуры представить имеющиеся данные, возникает довольно часто и его решение может ощутимо повлиять на эффективность и надежность предложенного решения. Даже несмотря на то, что многие алгоритмы и структуры данных уже реализованы в стандартных библиотеках, надо знать, когда и как ими нужно воспользоваться, их достоинства и недостатки — чтобы сделать правильный выбор и использовать весь спектр возможностей наиболее эффективно, а также при необходимости выполнить свою реализацию или внести изменения в существующую.

Чтобы уверенно работать с любым инструментом, нужно иметь не только знания, но и навык применения данного инструмента. Именно поэтому кроме теоретических сведений необходимо показать, как реализовать данный алгоритм (структуру данных) или как использовать существующие библиотеки на каком-то языке программирования.

Как выбрать язык программирования? Какой из них самый лучший? Понятно, что однозначного ответа на этот вопрос не существует.

C# — мощный объектно-ориентированный язык, наследник C/C++, который сохранил лучшие черты этих языков и вместе с тем стал проще и надежнее. Мощная библиотека каркаса .NET Framework поддерживает удобство построения различных типов приложений на языке C#, позволяя легко строить web-службы, другие виды компонентов, достаточно просто сохранять и получать информацию из базы данных и других хранилищ данных.

Данное пособие позволит сделать первый шаг в изучении языка C#, а также познакомиться с базовыми алгоритмами, которые применяются в частности при обработке текстовой и числовой информации, используются в таких областях как криптография, биоинформатика.

Еще один важный вопрос — чем данный учебник отличается от других, посвященных программированию на C#, структурам данных и алгоритмам?

Во-первых, для изучения данного учебника не нужны никакие предварительные знания по программированию, его может изучать любой человек, знакомый с компьютером на уровне пользователя. Простота изложения материала и большое количество разобранных примеров делают изучение языка C# доступным для широкого круга читателей.

Во-вторых, структура учебника такова, что каждая его глава содержит:

- теоретический материал;
- примеры решения типовых задач (учебных и практико-ориентированных);
- набор упражнений, рассчитанный на группу обучающихся из 15–20 человек и предназначенный для закрепления теоретического материала;
- задания для самостоятельной работы, что позволяет организовать частично-поисковую, исследовательскую и творческую работу обучающихся.

В-третьих, учебник реализует идеологию обучения программированию через изучение алгоритмов. Ведь именно знание алгоритмов закладывает фундамент профессионального программирования.

В-четвертых, учебник разрабатывался с учетом федеральных государственных образовательных стандартов высшего образования по IT-направлениям и IT-специальностям. Поэтому данный учебник может быть рекомендован:

- преподавателям вузов для подготовки и проведения занятий по таким учебным дисциплинам как «Информатика и программирование», «Структуры данных и алгоритмы», «Алгоритмы и их анализ» и т. д.;
- студентам IT-направлений и IT-специальностей начальных курсов для подготовки по соответствующим учебным дисциплинам;
- а также всем заинтересованным лицам, изучающим язык C# самостоятельно.

Данный учебник может использоваться и при обучении программированию в средней школе в рамках учебной дисциплины «Информатика и информационно-коммуникационные технологии» (профильный уровень), а также факультативных дисциплин и кружков по программированию.

Содержание и методика изложения учебного материала были апробированы, а затем внедрены в учебный процесс в рамках преподавания таких дисциплин как «Информатика и программирование», «Структуры данных и алгоритмы» при подготовке студентов младших курсов по различным IT-направлениям и IT-специальностям Федерального государственного бюджетного образовательного учреждения высшего образования «Саратовский национальный исследовательский государственный университет имени Н. Г. Чернышевского» (факультет компьютерных наук и информационных технологий). Следует отметить, что студенты данного вуза стабильно занимают высокие места на региональных, всероссийских и международных олимпиадах по программированию, а выпускники востребованы на рынке труда.

В результате изучения материала учебника обучающийся будет:

знать

- основные сведения о языке C#;
- специфику организации программного ввода-вывода данных на языке C#;

- способы хранения данных в программе через использование базовых типов и типов, объявленных пользователем, а также размерных и ссылочных типов;
- некоторые методы анализа алгоритмов;
- базовые алгоритмы работы с числами и последовательностями чисел;
- алгоритмы работы со строками;
- алгоритмы сортировки и поиска, их сравнительную характеристику и способы реализации на языке C#;

уметь

- использовать базовые типы данных и стандартные классы языка C# для решения учебных и практико-ориентированных задач;
- разрабатывать собственные типы данных на языке C# для решения практико-ориентированных задач;
- выполнять сравнительный анализ и осуществлять выбор подходящего алгоритма для решения учебных и практико-ориентированных задач;

владеть

- навыками реализации и сравнительного анализа алгоритмов;
- навыками практического программирования на языке C#.

Слова благодарности

Авторы благодарят ведущих сотрудников компании EPAM Systems Павла Агурова и Александра Кузнецова, оказавших неоценимый вклад в работу над данным учебником, за ценные критические замечания и советы по использованию языка C# для решения практико-ориентированных задач.

Авторы выражают искреннюю благодарность декану факультета компьютерных наук и информационных технологий федерального государственного бюджетного образовательного учреждения высшего образования «Саратовский национальный исследовательский государственный университет имени Н. Г. Чернышевского» (СГУ), кандидату физико-математических наук, доценту, лауреату премии Президента РФ в области образования, почетному работнику высшего образования Федоровой Антонине Гавриловне. Ваша поддержка, помощь и критические замечания сыграли большую роль не только в работе над данным учебником, но и в нашем профессиональном развитии.

Также авторы благодарят всех сотрудников кафедры информатики и программирования СГУ, принимавших участие в апробации содержания и методики изложения материала учебника во время проведения лекционных, практических и семинарских занятий со студентами факультета компьютерных наук и информационных технологий СГУ.

Глава 1
ВВЕДЕНИЕ

1.1. Платформа .NET, ее назначение и структура. Обзор технологий .NET

В 2000 г. компания Microsoft объявила о создании нового языка программирования — языка C#. Эта акция стала частью более значительного события — объявления о платформе .NET (.NET Framework), которая по сути представляла собой новую модель создания приложений, включающую в себя следующие возможности:

- использование библиотеки базовых классов, предлагающих целостную объектно-ориентированную модель программирования для всех языков программирования, поддерживающих .NET;
- полное и абсолютное межъязыковое взаимодействие, позволяющее разрабатывать фрагменты одного и того же проекта на различных языках программирования;
- общую среду выполнения приложений .NET, независимо от того, на каких языках программирования для данной платформы они были созданы; при этом среда берет на себя контроль за безопасностью выполнения приложений и управление ресурсами;
- упрощенный процесс развертывания приложения, в результате чего установка приложения может свестись к простому копированию файлов приложения в определенный каталог.

Одним из основных элементов .NET Framework является библиотека классов под общим именем FCL (*Framework Class Library*), к которой можно обращаться из различных языков программирования, в частности, из C#. Эта библиотека разбита на модули таким образом, что имеется возможность использовать ту или иную ее часть в зависимости от требуемых результатов. Так, например, в одном из модулей содержатся «кирпичики», из которых можно построить Windows-приложения, в другом — «кирпичики», необходимые для организации работы в сети и т. д.

Кроме FCL в состав платформы .NET входит единая среда выполнения программ CLR (*Common Language Runtime*), название которой говорит само за себя — это среда ответственна за поддержку выполнения всех типов приложений, разработанных на различных языках программирования с использованием библиотек .NET.

Замечание. Следует отметить, что основными языками, предназначенными для платформы .NET Framework, изначально являлись C#, VB.NET, Managed C++ и JScript.NET. Для данных языков Microsoft разработала компиляторы, переводящие программу в специальный код, называемый IL-кодом, который выполняется средой CLR. Кроме Microsoft еще несколько компаний и академических организаций создали свои собственные компиляторы, генерирующие код, работающий в CLR. На сегодняшний момент известны компиляторы для Pascal, Cobol, Lisp, Perl, Prolog и т. д. Это означает, что можно написать программу, например, на языке Pascal, а затем, воспользовавшись соответствующим компилятором, создать специальный код, который будет работать в среде CLR. Процесс компиляции и выполнения программы в среде CLR более подробно будет рассмотрен позже.

Среда CLR берет на себя всю низкоуровневую работу, например, автоматическое управление памятью. В языках программирования предыдущих поколений управление ресурсами, в частности, управление памятью, являлось одной из важных проблем. Объекты, созданные в памяти, должны быть удалены из нее, иначе в какой-то момент времени память будет исчерпана и программа не сможет продолжить выполнение. Так как это достаточно сложный процесс, часто программисты просто забывали удалять неиспользуемые объекты. При разработке платформы .NET эту проблему постарались решить. Теперь управление памятью берет на себя среда CLR. В процессе работы программы среда следит за объектами и автоматически уничтожает неиспользуемые.

Замечание. Система управления памятью называется сборщиком мусора GC (*Garbage Collector*).

Среда CLR обеспечивает интеграцию языков и позволяет объектам, созданным на одном языке, использовать объекты, написанные на другом. Такая интеграция возможна благодаря стандартному набору типов и информации, описывающей тип (метаданным). Интеграция языков очень сложная задача, так как некоторые языки не учитывают регистры символов, другие не поддерживают методы с переменным числом параметров и т. д. Чтобы создать тип, доступный для других языков, придется задействовать лишь те возможности языка, которые гарантированно доступны в других языках. С этой целью Microsoft разработал:

- общую систему типов CTS (*Common Type System*), которая описывает все базовые типы данных, поддерживаемые средой CLR, и определяет, как эти типы будут представлены в формате метаданных .NET;
- общеязыковую спецификацию CLS (*Common Language Specification*), описывающую минимальный набор возможностей, который должен быть реализован производителями компиляторов, чтобы их продукты работали в CLR, а также определяющую правила, которым

должны соответствовать видимые извне типы, чтобы к ним можно было получить доступ из любых других CLS-совместимых языков программирования.

Важно понимать, что система CLR/CTS поддерживает гораздо больше возможностей для программиста, чем спецификации CLS. Если при разработке какого-то типа требуется, чтобы он был доступен другим языкам, нельзя использовать возможности своего языка, выходящие за рамки возможностей, определяемых CLS. Иначе созданный тип окажется недоступным программистам, пишущим код на других языках. Если межъязыковое взаимодействие не требуется, то можно разрабатывать очень мощные типы, ограничиваясь лишь возможностями языка.

1.2. Принцип компиляции и выполнения программы в среде CLR. Управляемый и неуправляемый код

Создание приложений с помощью .NET Framework означает написание программы на любом языке программирования, который поддерживается этой платформой. Для того чтобы написанная, например на C#, программа была выполнена, ее необходимо преобразовать в программу на языке, «понятном» компьютеру (исполняемый/машинный код). Такой процесс преобразования называется компиляцией, а программа, которая его выполняет — компилятором. В прошлом почти все компиляторы генерировали код для конкретных процессорных архитектур. При разработке платформы .NET от этой зависимости постарались избавиться. Для этого ввели двухшаговую компиляцию.

На первом этапе все .NET компиляторы генерируют код на промежуточном языке IL (*Intermediate Language*) или IL-код. Другими словами компиляция со всех языков программирования .NET, включая C#, происходит в IL-код, который не является специфическим ни для какой операционной системы и ни для какого языка программирования. IL-код может быть выполнен в любой вычислительной системе, для которой реализована среда CLR.

На втором этапе IL-код переводится в код, специфичный для конкретной операционной системы и архитектуры процессора. Эта работа возлагается на JIT-компилятор (*Just In Time compiler* — компилирование точно к нужному моменту). Только после этого операционная система может выполнить приложение. Следует отметить, что JIT-компилятор входит в состав среды CLR.

IL-код, выполняемый под управлением CLR, называется управляемым (*managed*). Это означает, что среда CLR полностью управляет жизненным циклом программы: отслеживает безопасность выполнения команд программы, управляет памятью и т. д. Это несомненно является достоинством управляемого кода. Конечно, использовать приложения, разработанные на основе управляемого кода, можно только тогда, когда на компьютере установлена .NET Framework.

Использование IL-кода имеет и оборотную сторону — поддержка любой платформы означает отказ от функциональности, специфичной для конкретной платформы. Обойти это ограничение позволяет использование неуправляемого кода (*unmanaged*), т. е. кода, который не контролируется CLR или выполняется самой операционной системой. Такой код приходится использовать при необходимости обращения к низкоуровневым функциям операционной системы (например, понятие реестра существует только в Windows, и функции, работающие с реестром, приходится вызывать из операционной системы). Иногда использование неуправляемого кода позволяет ускорить выполнение некоторых алгоритмов.

1.3. Назначение и возможности Visual Studio.NET

В общем случае создавать файлы с исходным кодом на языке C# возможно с помощью обычного текстового редактора, например, Блокнота. Затем необходимо будет скомпилировать их в управляемый код через командную строку. Однако наиболее удобно для этих целей использовать среду Visual Studio.NET (VS), потому что:

— VS автоматически выполняет все шаги, необходимые для создания IL-кода;

— текстовый редактор VS изначально настроен для работы с теми .NET-языками, которые были разработаны Microsoft, в том числе C#, поэтому он может интеллектуально обнаруживать ошибки и «подсказывать» в процессе ввода, какой именно код можно использовать на данном этапе разработки (технология *IntelliSense*).

В состав VS входят средства, позволяющие создавать Windows- и Web-приложения путем простого перетаскивания мышью элементов пользовательского интерфейса.

Многие типы проектов, создание которых возможно на C#, могут разрабатываться на основе готовых шаблонов проектов. Вместо того чтобы каждый раз начинать с нуля, VS позволяет использовать уже имеющиеся файлы с исходным кодом, что уменьшает временны́е затраты на создание проекта.

VS содержит множество встроенных инструментов, облегчающих программисту процесс разработки приложений. Например, VS содержит встроенные средства профилирования, позволяющие выполнять тестирование приложения, определить проблемы его производительности на уровне исходного кода и многое другое.

Перечислим некоторые типы приложений, которые позволяет создавать VS.

Console Application — позволяют выполнять вывод на «консоль», то есть в окно командного процессора. Данный тип приложений существует со времен операционных систем с текстовым пользовательским интерфейсом, например MS-DOS. Однако консольные приложения продолжают активно использоваться и в наши дни. Их применение может

быть связано с отсутствием необходимости в графическом интерфейсе. Например, утилиты автоматической компиляции приложений, как правило, выполняются в заранее назначенные интервалы времени без участия пользователя (рис.1.1).

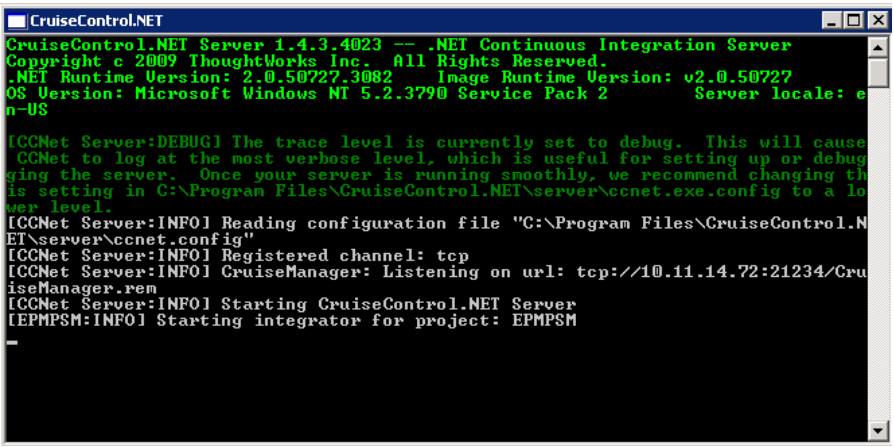


Рис. 1.1. **Процедура запуска системы контроля кода проекта**

Еще одним вариантом применения консольного ввода/вывода является встраивание его в программы с графическим интерфейсом. Дело в том, что современные программы содержат очень большое число команд, значительная часть которых никогда не используется обычными пользователями. В то же время эти команды должны быть доступны в случае необходимости. Ярким примером использования данного подхода являются компьютерные игры (рис. 1.2).

Рис. 1.2. **Пример использования консоли в игре**

Windows Forms Application — используют элементы графического интерфейса, включая формы, кнопки, флажки, переключатели и т. д. Приложе-

ния такого типа более удобны для пользователя, так как позволяют ему отдавать команды щелчком мыши, а не ручным вводом команд, что позволяет значительно повысить скорость работы по сравнению с консольными приложениями. Типичным примером приложения, построенного с применением графического интерфейса, является MS Word (рис. 1.3).

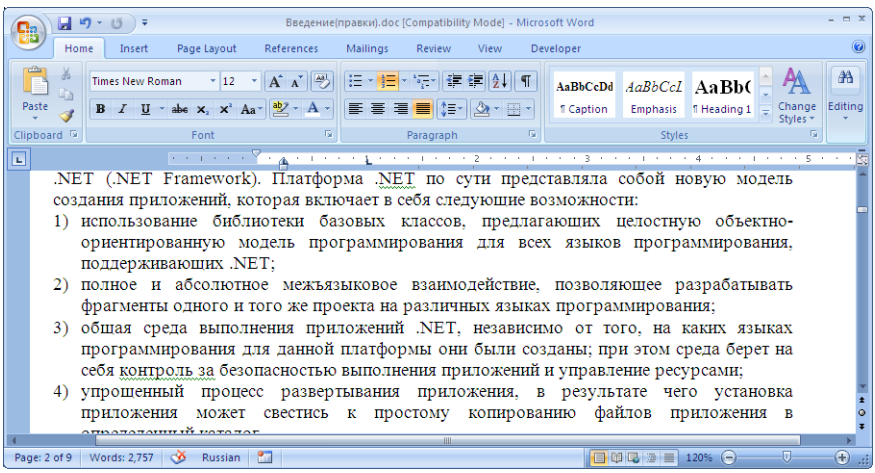


Рис. 1.3. **Графический интерфейс пользователя на примере MS Word**

Web Application — представляют собой набор web-страниц, которые могут просматриваться любым web-браузером. Web-приложения позволяют создавать многопользовательские системы с единообразным интерфейсом, практически не зависящим от установленных у пользователя операционной системы и остальных программ. Пример типичного web-приложения представлен на рис. 1.4.

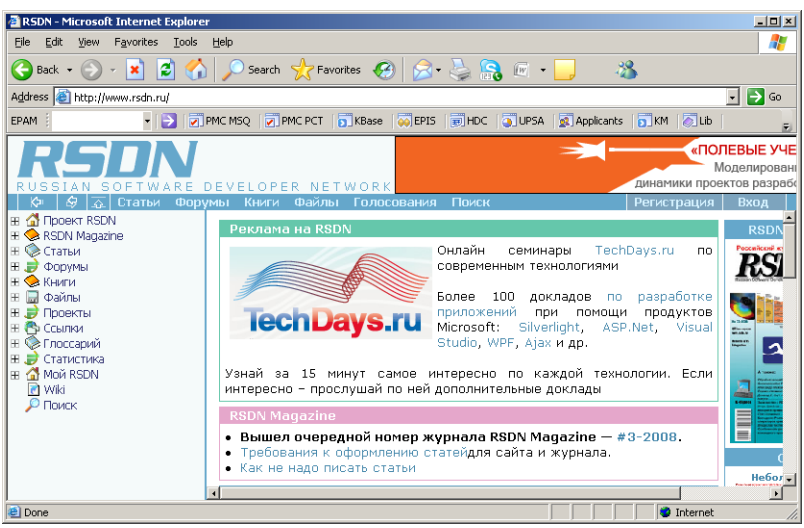


Рис. 1.4. **Пример web-приложения**

Web Service — представляют собой специализированные web-приложения, предназначенные для предоставления другим приложениям доступа к своим данным. В сети Интернет представлено довольно много различных сервисов: прогнозы погоды, котировки валют и др. Использование сервисов позволяет с минимальными затратами получать доступ к актуальной информации, которой владеют люди, не имеющие ни малейшего отношения к разрабатываемой нами программе.

Class Library — представляют собой библиотеки, содержащие классы и методы. Библиотеки не являются полноценным самостоятельными приложениями, но могут использоваться в других программах. Как правило, в библиотеки помещают алгоритмы и структуры данных, которые могут быть полезны более чем одному приложению.

Специфика платформы .NET такова, что она «подходит» для разработки «офисных» приложений, web-приложений, сетевых приложений и приложений для мобильных устройств. В то же время она не предназначена для создания операционных систем и драйверов.

В рамках данного учебника мы рассмотрим основы программирования на языке C#, разрабатывая консольные приложения в среде Visual Studio.NET, а также разберем базовые алгоритмы, которые широко используются при решении различных задач.

1.4. Создание первого проекта в среде Visual Studio

Приложение, находящееся в процессе разработки, называется проектом (*project*). Несколько проектов могут быть объединены в решение (*solution*). Совсем не обязательно, чтобы проекты в решении были одного типа. Например, в одно решение могут быть объединены консольные приложения, web-приложения и библиотеки, которые они используют. Мы будем создавать решения, состоящие из одного консольного приложения, так как консольные приложение самые простые по своей структуре и наиболее подходят для изучения основ программирования на языке C#.

Для создания проекта следует запустить VS. Пред вами откроется начальное окно VS (рис. 1.5).

Замечание. В зависимости от версии VS и от того, как в данный момент настроена среда, вид экрана может немного отличаться. Здесь и далее мы приводим примеры с использованием Microsoft Visual Studio Community 2017 (версия 15.7.5).

Затем на начальной странице VS (см. рис. 1.5) необходимо нажать ссылку *«Создать проект…»* или в главном меню выбрать команду *Файл → Создать проект…*. После этого откроется диалоговое меню *«Создать проект»* (рис. 1.6).

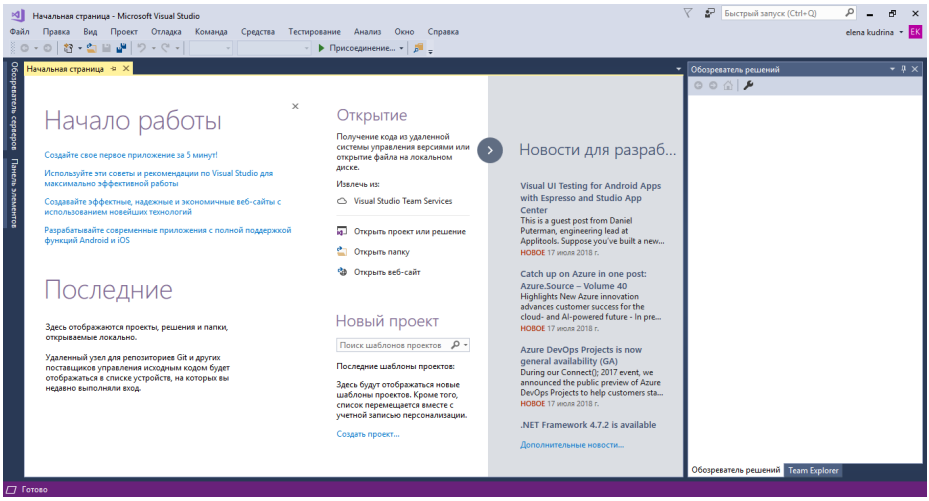


Рис. 1.5. **Начальная страница VS**

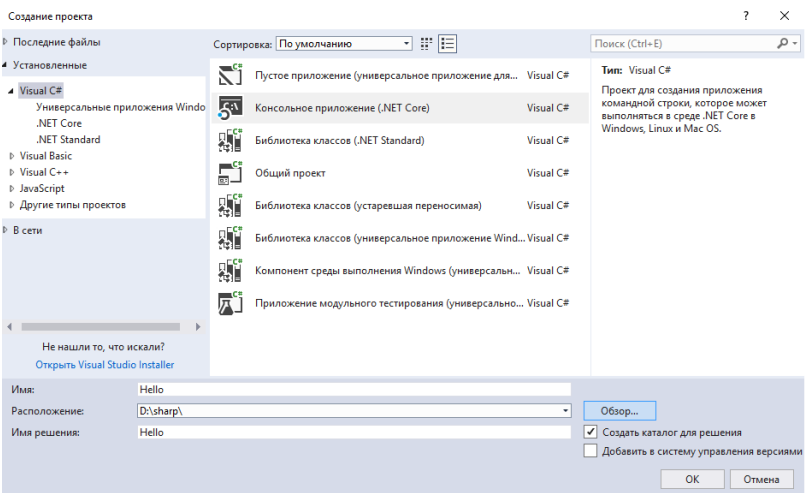


Рис. 1.6. **Диалоговое окно «Создать проект»**

Далее следует выбрать тип создаваемого приложения — *Консольное приложение*. В строчке *Имя* следует ввести новое имя приложения *Hello*. Расположение проекта зависит от установок, которые можно изменить, используя кнопку *Обзор*. Мы будем сохранять проекты на диск D в папку *sharp*. После проведенных установок необходимо нажать кнопку *OK*. Экран примет вид, изображенный на рис. 1.7.

В правой части экрана располагается окно управления проектом *Обозреватель решений*. В данном окне перечислены все ресурсы, входящие в проект. Пока нас будет интересовать только файл *Program.cs*, содержащий исходный код программы на языке C#. Если вы случайно закроете данное окно, то его можно включить командой *Вид* → *Обозреватель решений*.

В левой нижней части экрана VS обычно располагается окно *Список ошибок,* которое во время отладки проекта позволит получать информацию о локализации и типе ошибок. Если она отключено, как у нас (рис. 1.7), то его можно включить командой *Вид → Список ошибок.*

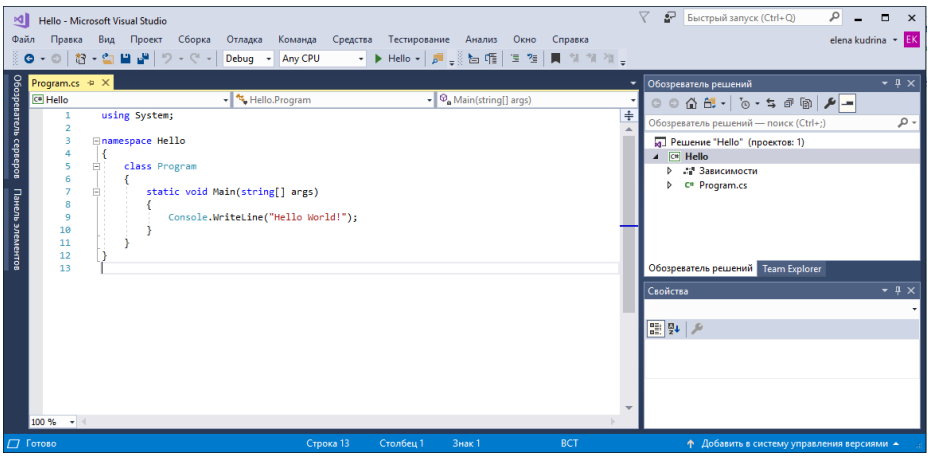


Рис. 1.7. **Рабочая область проекта Hello**

Основное пространство экрана занимает окно редактора, в котором располагается текст программы, созданный средой автоматически. Текст представляет собой каркас, в который программист будет добавлять нужный ему код и изменять его. При этом зарезервированные (ключевые) слова отображаются синим цветом, комментарии — зеленым, основной текст — черным. Текст структурирован. Щелкнув на знаке «минус» в первой колонке текста, мы скроем блок кода, щелкнув на знаке «плюс» — откроем.

Замечание. Однострочные комментарии начинаются с символов «//». Многострочные комментарии ограничиваются символами «/*» и «*/». Комментарии компилятором игнорируются и используются для документирования кода программы.

Теперь рассмотрим сам текст программы.

using System — это директива, которая разрешает использовать имена стандартных классов из пространства имен *System* непосредственно, без указания имени пространства, в котором они были определены. Так, например, если бы этой директивы не было, то нам пришлось писать бы *System.Console.WriteLine* (назначение данной команды мы рассмотрим позже). Конечно, писать полное пространство имен каждый раз очень неудобно. При указании директивы using можно писать просто имя, например, *Console.WriteLine.*

Для консольных программ ключевое слово *namespace* создает свое собственное пространство имен, которое по умолчанию называется

именем проекта. В нашем случае пространство имен называется *Hello*. Однако программист вправе указать другое имя.

Каждое имя, которое встречается в программе, должно быть уникальным. В больших и сложных приложениях используются библиотеки разных производителей. В этом случае трудно избежать конфликта между используемыми в них именами. Пространства имен предоставляют простой механизм предотвращения конфликтов имен. Они создают «подпространства» в глобальном пространстве имен.

C# — объектно-ориентированный язык, поэтому написанная на нем программа будет представлять собой совокупность взаимодействующих между собой классов. Автоматически был создан класс с именем *Program*. Данный класс содержит только один метод — метод *Main(),* который является точкой входа в программу. Это означает, что именно с данного метода начнется выполнение приложения. Каждая консольная программа на языке C# должна иметь метод *Main()*.

Перед объявлением типа возвращаемого значения для метода *Main* указан тип *void*, который означает, что данный метод не возвращает значение, а ключевое слово *static* означает, что метод *Main()* можно вызывать, не создавая экземпляр класса типа Program.

После имени метода в круглых скобках могут быть указаны параметры, передаваемые в метод *Main* при запуске программы. На данном этапе передачу параметров в метод *Main* мы проводить не будем, поэтому данный фрагмент кода *string [] args* можно удалить.

Тело метода *Main()* ограничивают парные фигурные скобки.

Обратите внимание на то, что Microsoft Visual Studio Community 2017 автоматически добавляет в текст программы команду:

```
Console.WriteLine("Hello World!");
```

Здесь *Console* — имя стандартного класса из пространства имен *System*, отвечающего за базовую систему ввода/вывода данных, т. е. клавиатуру и экран. Его метод *WriteLine* выводит на экран текст, заданный в кавычках. Следует отметить, что в более ранних версиях VS при создании проекта тело метода *Main* является пустым.

Чтобы запустить приложение, нужно нажать на кнопку *Start* (▶) стандартной панели инструментов VS или нажать кнопку F5. В результате программа скомпилируется в IL-код и этот код будет передан CLR на выполнение. Программа запустится, выведет на экран сообщение «Hello World!» и тут же завершит свою работу. Визуально это будет выглядеть как быстро мелькнувший на экране черный прямоугольник. Чтобы просмотреть сообщение в нормальном режиме, нужно нажать клавиши Ctrl+F5. В этом случае на экране будет отображено окно консольного приложения (рис. 1.8).

Замечание. В более сложных проектах имеет смысл явно вставить в конец программы команду Console.ReadLine(), позволяющую подождать до тех пор, пока пользователь не нажмет на клавишу Enter. Данный вариант является более предпочтительным, потому что нажатие Ctrl+F5 приводит к запуску приложения с отключенным режимом отладки, который довольно часто бывает необходим для поиска ошибок.

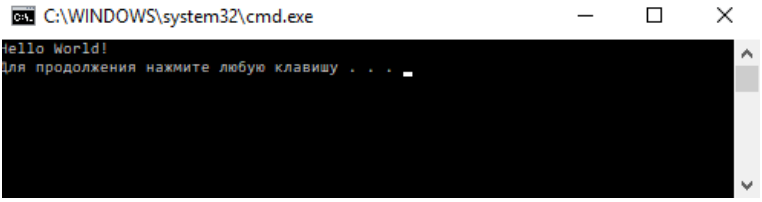


Рис. 1.8. **Окно консольного приложения**

Откроем папку *sharp* на диске *D*, содержащую проект, и изучим ее структуру. На данном этапе особый интерес для нас будут представлять следующие файлы.

Hello.sln — основной файл, отвечающий за все решение. Если необходимо открыть решение для работы, то нужно выбрать именно этот файл. Остальные файлы откроются автоматически.

Данный файл помечен пиктограммой 

Hello\Program.cs — файл, в котором содержится исходный код, написанный на языке C#. Именно с этим файлом мы и будем непосредственно работать через VS.

Данный файл помечен пиктограммой 

Hello\bin\Debug\netcoreapp2.0\Hello.dll — файл с IL-кодом вашего консольного приложения. Фактически это библиотека DLL, которую можно выполнить, введя команду *dotnet Hello.dll* в командной строке (рис. 1.9). Результат работы приложения представлен на рис. 1.10.

Данный файл помечается пиктограммой  .

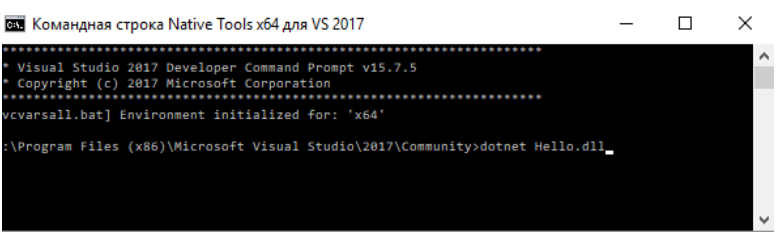


Рис. 1.9. **Запуск dll-файла из командной строки**

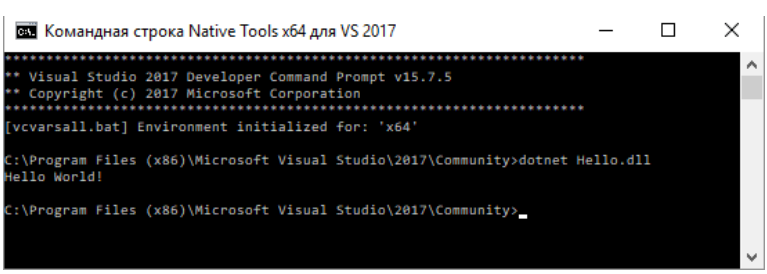


Рис. 1.10. **Результат работы консольного приложения, запущенного из командной строки.**

Следует отметить, что в рамках изучения основ программирования на языке C# с использованием VS нам не потребуется запускать приложения через командную строку, так как VS позволяет сделать это автоматически.

Замечания. Обратите внимание на то, что в папке netcoreapp2.0 вместе с файлом Hello.dll содержатся еще несколько взаимосвязанных файлов:

— Hello.deps.json — файл, содержащий все зависимости проекта;

— Hello.runtimeconfig.json — файл, хранящий информацию об общей среде выполнения, которую ожидает приложение, а также другие параметры конфигурации для среды выполнения (например, тип сборки мусора);

— Hello.pdb — файл базы данных программы или так называемый файл символов.

Эти файлы крайне важны для запуска Hello.dll, так как содержат служебную информацию.

Команда dotnet, как видно из рис. 1.9—1.10, запускает на выполнение соответствующие dll-файлы, хранящиеся в папке C:\Program Files (x86)\Microsoft Visual Studio\2017\Community. Если нужно запустить на выполнение файлы, располагающиеся в другом месте файловой системы, то нужно выполнить команду dotnet PathToFile\file.dll, где PathToFile — полный путь к dll-файлу.

Теперь рассмотрим, как работает технология IntelliSense, на примере добавления команды *Console.ReadLine()* в текст программы (рис. 1.11).

Уже в начале процесса ввода команды открывается контекстное меню, которое подсказывает все «знакомые» слова, начинающиеся на букву C. Когда будет введено словосочетания «Con», контекстное меню высветит слово *Console*, а также выведет справочную информацию о данном классе. Если теперь вы нажмете на клавишу *Enter*, то данное слово автоматически вставится в текст программы.

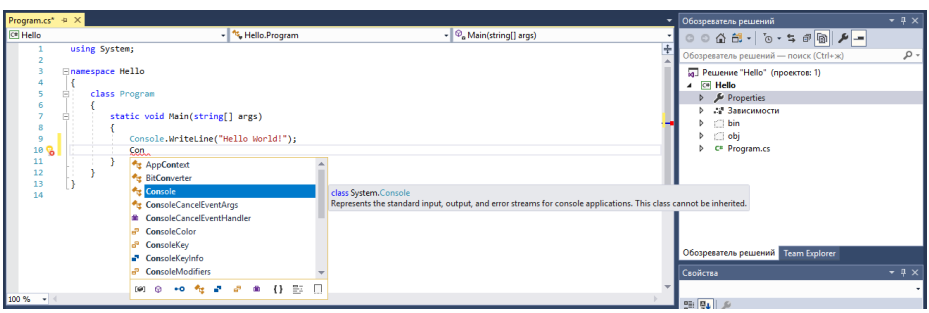


Рис. 1.11. **Технология IntelliSense**

Теперь поставьте точку и наберите букву R. Контекстное меню будет не только показывать все методы и свойства, начинающиеся на букву R, но и приводить справочную информацию о том, за что отвечает тот или иной метод или свойство класса.

Если вы щелкнете левой кнопкой мышки на имени метода *ReadLine* (рис. 1.12), то вы получите справочную информацию о данном методе, если дважды щелкнете левой кнопкой мышки, то имя метода автоматически добавится в текст программы.

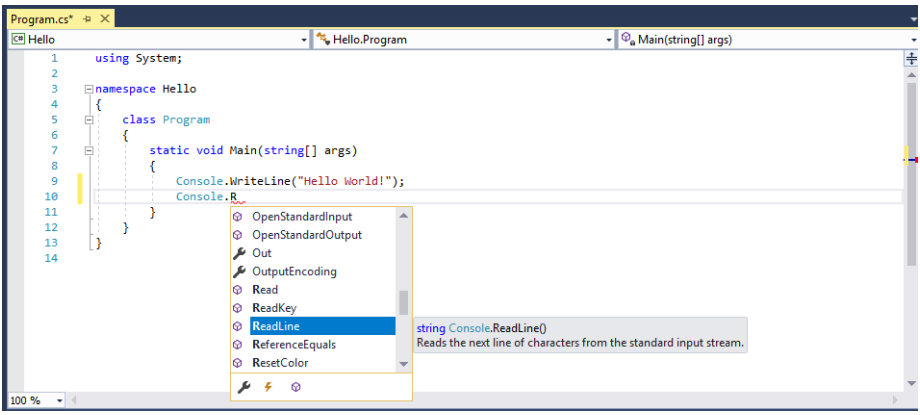


Рис. 1.12. **Получение справочной информации с использованием технологии IntelliSense**

Если код программы будет содержать ошибки, например, пропущена точка с запятой после команды вывода, то IntelliSense подсветит ошибку, а в окне *Список ошибок* появится подробное описание ошибки (рис. 1.13).

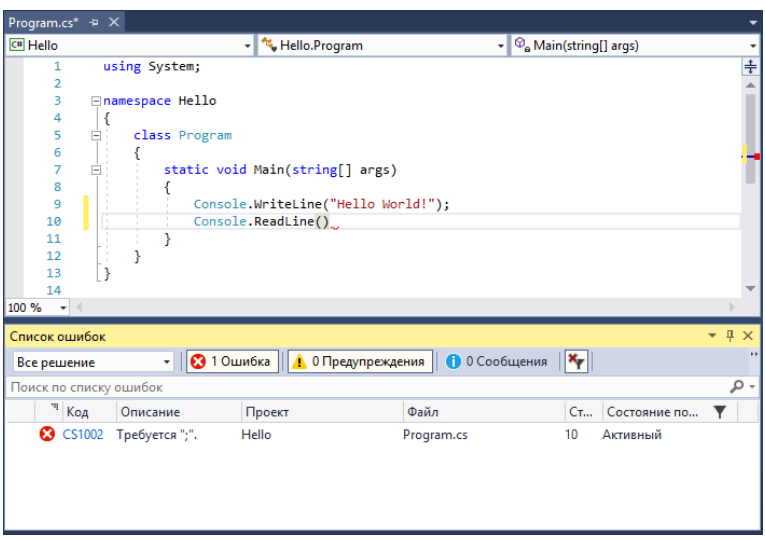


Рис. 1.13. **Информация об ошибках**

Обратите внимание на то, что в окне *Список ошибок* отображается информация не только о самих ошибках (критических ситуациях, не позволяющих сгенерировать IL-код), но и информация о предупреж-

дениях (ситуациях, потенциально опасных уже в процессе выполнения IL-кода).

Задания

1. Измените текст кода консольного приложения так, чтобы на экран выводилось сообщение «*Привет! Меня зовут Вася!*».

2. Изучите, чем метод Write отличается от метода WriteLine.

3. Попробуйте сделать несколько ошибок, например, не закрыть скобку или поставить лишнюю скобку, допустить ошибку в ключевых словах, и посмотрите, какие сообщения появятся в окне Список ошибок.

Самостоятельная работа

1. Познакомьтесь с программой Microsoft Imagine (https://imagine.microsoft.com/ru-ru) и узнайте, какие возможности она дает вам для изучения языка C# и VS.

2. Произведите установку необходимого программного обеспечения на свой персональный компьютер: https://imagine.microsoft.com/ru-ru/catalog.

3. Познакомьтесь с порталом разработчиков MSDN (http://msdn.microsoft.com/ru-RU/). Особое внимание следует уделить ресурсам, посвященным языку C# (https://docs.microsoft.com/ru-ru/dotnet/csharp/), а также стратегии развития Microsoft Visual Studio (https://docs.microsoft.com/ru-ru/visualstudio/productinfo/vs2018-roadmap).

Глава 2
БАЗОВЫЕ ЭЛЕМЕНТЫ ЯЗЫКА C#

2.1. Состав языка

Алфавит — совокупность допустимых в языке символов. Алфавит языка C# включает:

- прописные и строчные латинские буквы и буквы национальных алфавитов (включая кириллицу);
- арабские цифры от 0 до 9, шестнадцатеричные цифры от A до F;
- специальные знаки: " {}, | ; [] () + - / % *. \ ' :? < = >! & ~ ^ @ _
- пробельные символы: пробел, символ табуляции, символ перехода на новую строку.

Из символов алфавита формируются лексемы языка: идентификаторы, ключевые (зарезервированные) слова, знаки операций, константы, разделители (скобки, точка, запятая, пробельные символы). Границы лексем определяются другими лексемами, такими, как разделители или знаки операций. В свою очередь лексемы входят в состав выражений (выражение задает правило вычисления некоторого значения) и операторов (оператор задает законченное описание некоторого действия).

Идентификатор — это имя программного элемента: константы, переменной, метки, типа, класса, объекта, метода и т. д. Идентификатор может включать латинские буквы и буквы национальных алфавитов, цифры и символ подчеркивания. Прописные и строчные буквы различаются, например, myname, myName и MyName — три различных имени. Первым символом идентификатора может быть буква или знак подчеркивания, но не цифра. Пробелы и другие разделители внутри имен не допускаются. Язык C# не налагает никаких ограничений на длину имен, однако для удобства чтения и записи кода не стоит делать их слишком длинными.

Для улучшения читабельности кода программным элементам следует давать осмысленные имена, составленные в соответствии с определенными правилами. Существует несколько видов нотаций — соглашений о правилах создания имен.

В нотации Pascal каждое слово, входящее в идентификатор, начинается с заглавной буквы. Например:Age, LastName, TimeOfDeath.

Венгерская нотация отличается от предыдущей наличием префикса, соответствующего типу величины. Например: fAge, sName, iTime.

В нотации Camel с заглавной буквы начинается каждое слово идентификатора, кроме первого. Например: age, lastName, timeOfDeath.

Наиболее часто используются нотации Pascal или Camel. Однако в простых программах будут использоваться однобуквенные переменные.

Ключевые слова — это зарезервированные идентификаторы, которые имеют специальное значение для компилятора, например, static, int и т. д. Ключевые слова можно использовать только по прямому назначению. Однако если перед ключевым словом поставить символ @, например, @int, @static, то полученное имя можно использовать в качестве идентификатора. С полным перечнем ключевых слов и их назначением можно ознакомиться в справочной системе C#.

Замечание. Другие лексемы (знаки операций и константы), а также правила формирования выражений и различные виды операторов будут рассмотрены чуть позже.

2.2. Типы данных

C# является языком со строгой типизацией. В нем необходимо объявлять тип всех создаваемых программных элементов (например, переменных, объектов, окон, кнопок и т. д.), что позволяет среде CLR предотвращать возникновение ошибок, следя за тем, чтобы объектам присваивались значения только разрешенного типа. Тип программного элемента сообщает компилятору о его размере (например, тип *int* показывает, что объект занимает 4 байта) и возможностях (например, кнопка может быть нарисована, нажата и т. д.).

В C# типы делятся на три группы:

— *базовые* типы — предлагаемые языком;

— типы, *определяемые пользователем;*

— анонимные типы — типы, которые автоматически создаются на основе инициализаторов объектов (начиная с версии C# 3.0).

Кроме того, типы C# разбиваются на две другие категории: *типы значения* (*value type*) и *ссылочные типы* (*reference type*). Почти все базовые типы являются типами значениями. Исключение составляют типы *object* и *string,* которые являются базовыми, но ссылочными типами данных. Все пользовательские типы, кроме структур, являются ссылочными. Дополнительно к упомянутым типам, язык C# поддерживает типы *указателей,* однако они используются только с неуправляемым кодом.

Принципиальное различие между типами значениями и ссылочными типами состоит в способе хранения их значений в памяти. В пер-

вом случае фактическое значение хранится в стеке (или как часть большого объекта ссылочного типа). Адрес переменной ссылочного типа тоже хранится в стеке, но сам объект хранится в куче.

Стек — это структура, используемая для хранения элементов по принципу LIFO (*Last in — first out*, последним пришел — первым ушел). В данном случае под стеком понимается область памяти, обслуживаемая процессором, в которой хранятся значения локальных переменных. *Куча* — область памяти, используемая для хранения данных, работа с которыми реализуется через указатели и ссылки. Память для размещения таких данных выделяется программистом динамически, а освобождается сборщиком мусора.

Сборщик мусора уничтожает программные элементы в стеке через некоторое время после того, как закончит существование раздел стека, в котором они объявлены. То есть, если в пределах блока (фрагмента кода, помещенного в фигурные скобки {}) объявлена локальная переменная, соответствующий программный элемент будет удален по окончании работы данного блока. Объект в куче подвергается сборке мусора через некоторое время после того, как уничтожена последняя ссылка на него.

Язык C# предлагает обычный набор базовых типов (табл. 2.1), каждому из них соответствует тип, поддерживаемый общеязыковой спецификацией CLS платформы .NET.

Таблица 2.1

Базовые типы C#

Тип в языке C#	Размер в байтах	Тип .NET	Описание
object		Object	Может хранить все что угодно, так как является всеобщим предком
Логический тип			
bool	1	Boolean	true или false
Целые типы			
sbyte	1	SByte	Целое со знаком (от –128 до 127)
byte	1	Byte	Целое без знака (от 0 до 255)
short	2	Int16	Целое со знаком (от –32 768 до 32 767)
ushort	2	UInt16	Целое без знака (от 0 до 65 535)
int	4	Int32	Целое со знаком (от –2 147 483 648 до 2 147 483 647)
uint	4	UInt	Целое число без знака (от 0 до 4 294 967 295)
long	8	Int64	Целое со знаком (от –9 223 372 036 854 775 808 до 9 223 372 036 854 775 807)
ulong	8	UInt64	Целое без знака (от 0 до 0ffffffffffffff)

Окончание табл. 2.1

Тип в языке C#	Размер в байтах	Тип .NET	Описание
Вещественные типы			
float	4	Single	Число с плавающей точкой двойной точности. Содержит значения приблизительно от $\pm 1{,}5 \cdot 10^{-45}$ до $\pm 3{,}4 \cdot 10^{38}$ с 7 значащими цифрами
double	8	Double	Число с плавающей точкой двойной точности. Содержит значения приблизительно от $\pm 5{,}0 \cdot 10^{-324}$ до $\pm 1{,}7 \cdot 10^{308}$ с 15—16 значащими цифрами
Символьный тип			
char	2	Char	Символ Unicode
Строковый тип			
string		String	Строка из Unicode-символов
Финансовый тип			
decimal	12	Decimal	Число до 28 знаков с фиксированным положением десятичной точки. Обычно используется в финансовых расчетах.

2.3. Переменные и константы

Переменная представляет собой типизированную область памяти. Программист создает переменную, объявляя ее тип и указывая имя. При объявлении переменной ее можно инициализировать (присвоить ей начальное значение), а затем в любой момент ей можно присвоить новое значение, которое заменит собой предыдущее.

```
using System;
namespace MyProject
{
  class Program
  {
    static void Main()
    {

      int i=10;                    // объявление и инициализация
                                   // целочисленной переменной i
      Console.WriteLine(i);        // просмотр значения переменной
      i=100;                       // изменение значение переменной
      Console.WriteLine(i);        // вывод значения i на экран
    }
  }
}
```

Замечание. Здесь мы привели пример полного варианта программы. В дальнейшем мы будем приводить только фрагменты программы, например, только содержимое класса Program или содержимое метода Main.

В языках предыдущего поколения переменные можно было использовать без инициализации. Это могло привести к множеству проблем и долгому поиску ошибок. В языке C# требуется, чтобы переменные были явно проинициализированы до их использования. Проверим этот факт на примере.

```
static void Main()
{
  int i;  // объявление переменной без инициализации
  Console.WriteLine(i);  // просмотр значения переменной
}
```

При попытке скомпилировать этот пример в списке ошибок будет выведено следующее сообщение: *«Использование локальной переменной i, которой не присвоено значение»*.

Инициализировать каждую переменную сразу при объявлении не обязательно, но необходимо присвоить ей значение до того, как она будет использована.

Константа, в отличие от переменной, не может менять свое значение. Константы бывают трех видов: *литералы, типизированные константы* и *перечисления*.

В операторе присваивания

```
x = 32;
```

число 32 является *литеральной константой*. Его значение всегда равно 32 и его нельзя изменить.

Типизированные константы именуют постоянные значения. Объявление типизированной константы происходит следующим образом:

```
const <тип> <идентификатор> = <значение>;
```

Рассмотрим пример:

```
static void Main()
{
  const int i = 10;     // объявление целочисленной константы i
  Console.WriteLine(i);  // просмотр значения константы
  i = 100;  // ошибка – недопустимо изменять значение константы
  Console.WriteLine(i);
}
```

Задание. Измените программу так, чтобы при объявлении константы не происходила инициализация. Как на это отреагирует компилятор и почему?

Перечисления *(enumerations)* являются альтернативой константам. Перечисление — это особый размерный тип, состоящий из набора име-

нованных констант (называемых *списком перечисления)*. Синтаксис объявления перечисления следующий:

```
[атрибуты] [модификаторы] enum <имя> [: базовый тип]
{
  список-перечисления констант(через запятую)
};
```

Замечание. Атрибуты и модификаторы являются необязательными элементами этой конструкции. Более подробные сведения о них можно найти в дополнительных источниках информации.

Базовый тип — это тип самого перечисления. Если не указать базовый тип, то по умолчанию будет использован тип *int*. В качестве базового типа можно выбрать любой целый тип, кроме *char*. Пример использования перечисления:

```
class Program
{
  enum gradus:int
  {
    min=0,
    krit=72,
    max=100,  // 1
  }
  static void Main()
  {
    Console.WriteLine("минимальная температура=" +
      (int) gradus.min);
    Console.WriteLine("критическая температура=" +
      (int)gradus.krit);
    Console.WriteLine("максимальная температура=" +
      (int)gradus.max);
  }
}
```

Замечания. В общем случае последнюю запятую в объявлении перечисления можно не ставить (см. строку 1). Но лучше ее поставить: если вам придется добавить еще несколько строк в перечисление, такая предусмотрительность избавит вас от возможных синтаксических ошибок.

Запись (int) gradus.min используется для явного преобразования перечисления к целому типу. Если убрать (int), то на экран будет выводиться название констант.

Символ «+» в записи "минимальная температура = " + (int) gradus.min при обращении к методу WriteLine означает, что строка "минимальная температура = " будет «склеена» со строковым представлением значения (int) gradus.min. В результате получится новая строка, которая и будет выведена на экран.

2.4. Организация ввода-вывода данных. Форматирование

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 классы. В частности, для работы с консолью используется стандартный класс *Console*, определенный в пространстве имен *System*.

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. Рассмотрим их более подробно:

```
// На экран выводится значение идентификатора x
Console.WriteLine(x);
/* На экран выводится строка, образованная последовательным
слиянием строки "x=", значения x, строки "y=" и значения y */
Console.WriteLine("x=" + x +"y=" + y);
/* На экран выводится строка, формат которой задан
первым аргументом метода, при этом вместо параметра {0}
выводится значение x, а вместо {1} – значение y */
Console.WriteLine("x={0} y={1}", x, y);
```

Далее мы будем использовать только третий вариант, поэтому рассмотрим его более подробно. Пусть нам дан следующий фрагмент программы:

```
int i = 3, j = 4;
Console.WriteLine("{0} {1}", i, j);
```

При обращении к методу WriteLine через запятую перечисляются три аргумента: *"{0} {1}", i, j*. Первый аргумент *"{0} {1}"* определяет формат выходной строки. Следующие аргументы нумеруются с нуля, так переменная i имеет номер 0, j — номер 1. Значение переменной i будет помещено в выходную строку на место параметра {0}, а значение переменной j — на место параметра {1}. В результате на экран будет выведена строка: 3 4.

Если мы обратимся к методу WriteLine следующим образом:

```
Console.WriteLine("{0} {1} {0)", j, i);
```

то на экран будет выведена строка: 4 3 4.

Данный вариант использования метода *WriteLine* является наиболее универсальным, потому что он позволяет не только выводить данные на экран, но и управлять форматом их вывода. Рассмотрим несколько примеров.

1. Использование управляющих последовательностей. Управляющей последовательностью называют определенный символ, предва-

ряемый 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 (табл 2.2).

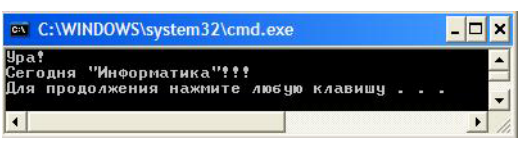
Таблица 2.2

Управляющие символы

Вид	Наименование
\a	Звуковой сигнал
\b	Возврат на шаг назад
\f	Перевод страницы
\n	Перевод строки
\r	Возврат каретки
\t	Горизонтальная табуляция
\v	Вертикальная табуляция
\\	Обратная косая черта
\'	Апостроф
\"	Кавычки

Пример:

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



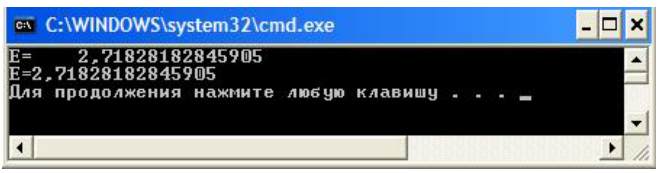
Замечание. Вместо управляющей последовательности \n можно использовать константу Enviropment.NewLine. Она более универсальна, так как ее значение зависит от контекста и операционной системы, в которой запускается программа.

Задание. Измените программу так, чтобы все сообщение выводилось в одну строку, а после вывода сообщения раздавался звуковой сигнал.

2. Управление размером поля вывода. Первым аргументом *WriteLine* указывается строка вида {n, m} — где n определяет номер идентификатора из списка аргументов метода *WriteLine*, а m — количество позиций (размер поля вывода), отводимых под значение данного

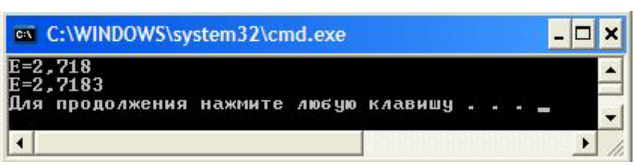
идентификатора. При этом значение идентификатора выравнивается по правому краю. Если выделенных позиций для размещения значения идентификатора окажется недостаточно, то автоматически добавится необходимое количество позиций. Пример:

```
static void Main()
{
  double x = Math.E;
  Console.WriteLine("E={0,20}", x);
  Console.WriteLine("E={0,10}", x);
}
```



3. Управление размещением вещественных данных. Первым аргументом *WriteLine* указывается строка вида *{n: ##.###}* — где *n* определяет номер идентификатора из списка аргументов метода *WriteLine,* а *##.###* определяет формат вывода вещественного числа. В данном случае под целую часть числа отводится две позиции, под дробную — три. Если выделенных позиций для размещения целой части значения идентификатора окажется недостаточно, то автоматически добавится необходимое количество позиций. Пример:

```
static void Main()
{
  double x= Math.E;
  Console.WriteLine("E={0:##.###}", x);
  Console.WriteLine("E={0:.####}", x);
}
```



Задание. Измените программу так, чтобы число e выводилось на экран с точностью до 6 знаков после запятой.

4. Управление форматом числовых данных. Первым аргументом *WriteLine* указывается строка вида *{n:<спецификатор>m}* — где n определяет номер идентификатора из списка аргументов метода *WriteLine, <спецификатор>* — определяет формат данных, а *m* — количество позиций для дробной части значения идентификатора. В качестве спецификаторов могут использоваться значения, приведенные в табл. 2.3.