

Visual C#[®] 2010



HOW TO PROGRAM

FOURTH EDITION



PAUL DEITEL
HARVEY DEITEL



Preface

Welcome to the Visual C#[®] 2010 programming language and the world of Microsoft[®] Windows[®] and Internet programming with Microsoft's .NET platform!

This book focuses on software engineering best practices. At the heart of the book is the Deitel signature “live-code approach.” Concepts are presented in the context of working programs, rather than in code snippets. Each code example is accompanied by sample executions. All the source code is available at www.deitel.com/books/vcsharp2010http/ and at the book's Companion Website at www.pearsonhighered.com/deitel/.

As you read the book, if you have questions, send an e-mail to deitel@deitel.com; we'll respond promptly. For updates on this book and its supporting Visual C# software, visit www.deitel.com/books/vcsharp2010http/, follow us on Twitter (@deitel) and Facebook (www.deitel.com/deite1fan), and subscribe to the *Deitel[®] Buzz Online* newsletter (www.deitel.com/newsletter/subscribe.html).

New and Updated Features

Here are the updates we've made for *Visual C#[®] 2010 How to Program, 4/e*:

- ***Printed book contains core content; advanced chapters are online.*** The printed book contains sufficient core content for most introductory Visual C# course sequences. Several online chapters are included for more advanced courses and for professionals. These are available in searchable PDF format on the book's password-protected Companion Website—see the access card in the front of this book.
- The book's Companion Website includes extensive *VideoNotes* in which co-author Paul Deitel explains in detail most of the programs in the core chapters.
- ***Making a Difference exercises set.*** We encourage you to use computers and the Internet to research and solve significant social problems. These new exercises are meant to increase awareness and discussion of important issues the world is facing. We hope you'll approach them with your own values, politics and beliefs.
- ***Up-to-date with Visual C# 2010, C# 4, the Visual Studio 2010 IDE and .NET 4.*** The C# language has been standardized internationally by ECMA and ISO. The latest version of that language is referred to as C# 4. Microsoft's implementation of this standard is referred to as Visual C# 2010.
- ***New language features.*** We cover new C# features, such as optional parameters, named parameters, covariance and contravariance.
- ***Databases.*** We use Microsoft's free SQL Server Express (which installs with the free Visual C# Express) to teach the fundamentals of database programming. Chapters 18, 19, 27 and 28 use database and LINQ fundamentals in the context of an address-book desktop application, a web-based guestbook, a bookstore and an airline reservation system.

- *ASP.NET 4.* Microsoft's .NET server-side technology, ASP.NET, enables you to create robust, scalable web-based applications. In Chapter 19, you'll build several applications, including a web-based guestbook application that uses ASP.NET, LINQ and a `DataSource` to store data in a database and display data in a web page. The chapter also discusses the ASP.NET Development Server for testing your web applications on your local computer.
- *We removed generic methods* from Chapter 9 to make the code easier to understand.
- The code will run on *Windows 7*, *Windows Vista* and *Windows XP*. We'll post any issues on www.deitel.com/books/vcsharp2010http/.
- *We introduce exception handling much earlier* (Chapter 8) and integrated it in subsequent chapters in which it had not been used previously. We also now throw exceptions for invalid data received in the set accessors of properties.
- *New design.* The book has a new interior design that graphically organizes, clarifies and highlights the information, and enhances the book's pedagogy. We used italics extensively to emphasize important words, phrases and points in the text.
- *We titled the programming exercises* to help instructors tailor assignments.

Other features of *Visual C# 2010 How to Program, 4/e* include:

- We've provide instructors with *solutions to the vast majority of the exercises*. There are a few large exercises marked "Project" for which solutions are not provided.
- *We use LINQ (Language Integrated Query) to query files, databases, XML and collections.* The introductory LINQ chapter, Chapter 9, in the core printed book is intentionally brief to encourage instructors to cover this important technology early. The online chapters continue the discussion of LINQ.
- *Local type inference.* When you initialize a local variable in its declaration, you can now omit the variable's type—the compiler infers it from the initializer value.
- *Object initializers.* For new objects, you can use object initializer syntax (similar to array initializer syntax) to assign values to the new object's `public` properties and `public` instance variables.
- *We emphasize the IDE's IntelliSense feature* that helps you write code faster and with fewer errors.

Our Text + Digital Approach to Content

We surveyed hundreds of instructors teaching Visual C# courses and learned that most want a book with content focused on their introductory courses. With that in mind, we moved various intermediate and advanced chapters to the web. Having this content in digital format makes it easily searchable, and gives us the ability to fix errata and add new content as appropriate. The book's Companion Website at

www.pearsonhighered.com/deitel/

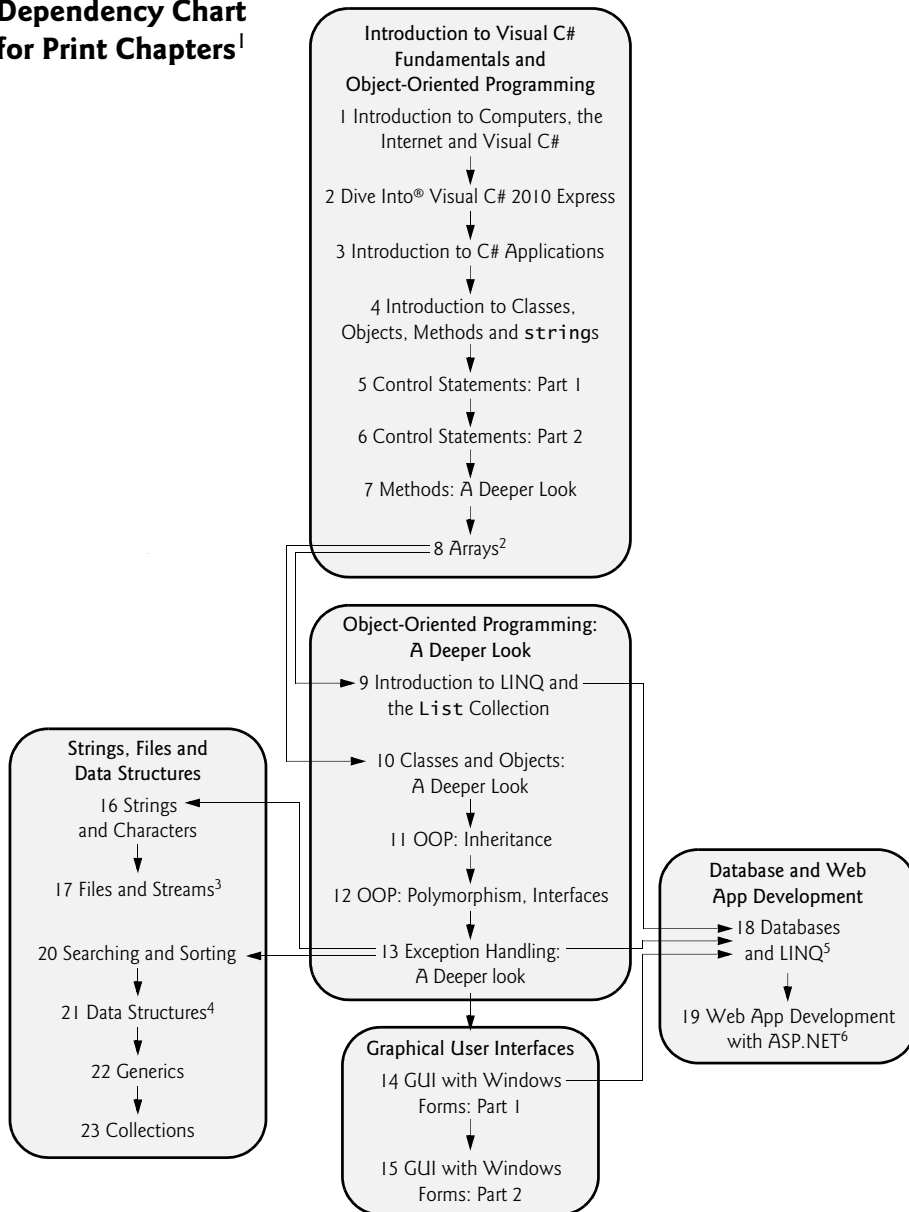
(see the access card at the front of the book) contains the following chapters in *searchable* PDF format:

- *WPF (Windows Presentation Foundation) GUI, graphics and multimedia.* We extend the core book's GUI coverage in Chapters 24–25 with an introduction to Windows Presentation Foundation (WPF)—Microsoft's new framework that integrates GUI, graphics and multimedia capabilities. We implement a painting application, a text editor, a color chooser, a book-cover viewer, a television video player, various animations, and speech synthesis and recognition applications.
- *ASP.NET 4 and ASP.NET AJAX.* Chapter 27 extends Chapter 19's ASP.NET discussion with a case study on building a password-protected, web-based bookstore application. We also introduce ASP.NET AJAX controls and use them to add AJAX functionality to web applications to improve their responsiveness.
- *WCF (Windows Communication Foundation) Web Services.* Web services enable you to package application functionality in a manner that turns the web into a library of reusable services. In Chapter 28, we include case studies on building an airline reservation web service, a blackjack web service and a math question generator web service that's called by a math tutor application.
- *Silverlight.* Chapter 29 introduces Silverlight, which enables you to create visually stunning, multimedia-intensive user interfaces for web applications. The chapter presents powerful multimedia applications, including a weather viewer, Flickr photo viewer, deep zoom book-cover collage and video viewer.
- *Visual C# XML capabilities.* Use of the Extensible Markup Language (XML) is exploding in the software-development industry and in e-business, and is pervasive throughout the .NET platform. In Chapter 26, we use show how to programmatically manipulate the elements of an XML document using LINQ to XML.
- *Optional Case Study: Using the UML to Develop an Object-Oriented Design and C# Implementation of an ATM.* The UML™ (Unified Modeling Language™) is the preferred graphical modeling language for designing object-oriented systems. This edition includes an optional online case study on object-oriented design using the UML (Chapters 30–31). We design and implement the software for a simple automated teller machine (ATM). We analyze a typical requirements document that specifies the system to be built. We determine the classes needed to implement that system, the attributes the classes need to have, the behaviors the classes need to exhibit and specify how the classes must interact with one another to meet the system requirements. From the design we produce a working Visual C# implementation. We've presented this case study to professional audiences in C#, Java, Visual Basic and C++. After seeing the case-study presentation, students report having a “light-bulb moment”—the case study “ties it all together” for them and helps them understand how objects in a larger system communicate with one another.
- *Index.* The online index includes the content from the printed book *and* the online content. The printed book index covers only the printed material.

Dependency Charts

The charts in Figs. 1–2 show the dependencies among the chapters to help instructors plan their syllabi. The printed book focuses on introductory course sequences (Fig. 1). The online chapters include intermediate and advanced content for more advanced courses (Fig. 2).

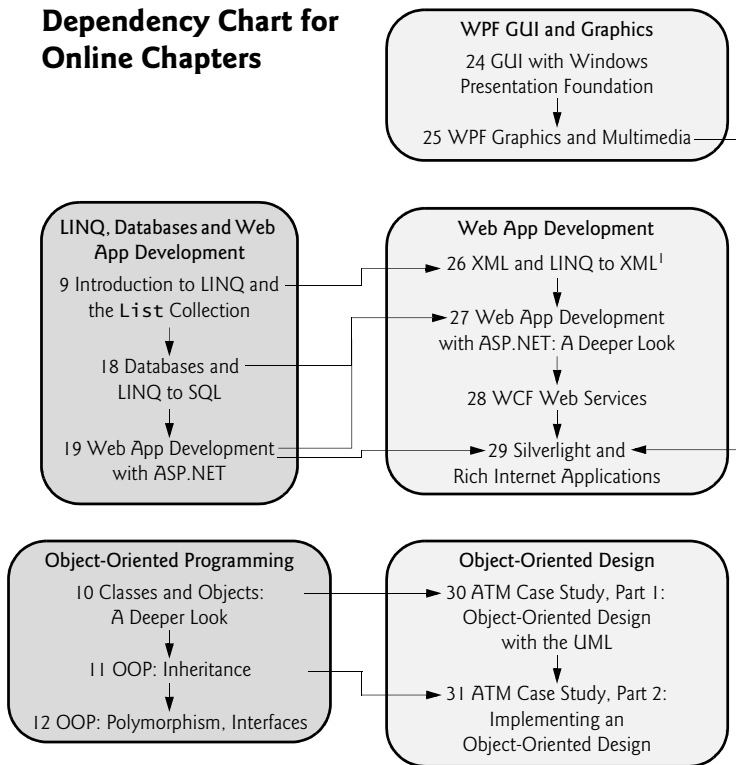
Dependency Chart for Print Chapters¹



1. See Fig. 2 for the online chapters.
2. Chapter 8 introduces exception handling.
3. Requires Sections 14.1–14.5.
4. Requires Sections 14.1–14.5 and 15.6.
5. Requires Sections 14.1–14.6 and 15.8.
6. Requires general GUI and event-handling knowledge (Sections 14.1–14.3).

Fig. 1 | Chapter dependency chart for the chapters in the printed book.

Dependency Chart for Online Chapters



1. Chapter 26 depends on the introduction to XML in Chapter 24.

Fig. 2 | Chapter dependency chart for the online chapters.

Teaching Approach

Visual C# 2010 How to Program, 4/e contains a rich collection of examples. We concentrate on building good software and stress program clarity.

Live-Code Approach. The book is loaded with “live-code” examples. Most new concepts are presented in the context of complete working Visual C# applications, followed by one or more executions showing program inputs and outputs. In the few cases where we use snippets, we tested them in complete working programs then copied the code from the program and pasted it into the book.

Syntax Shading. For readability, we syntax shade the code, similar to the way most integrated-development environments and code editors syntax color the code. Our syntax-shading conventions are:

```

comments appear like this
keywords appear like this
constants and literal values appear like this
all other code appears in black
  
```

Code Highlighting. We place gray rectangles around each program’s key code.

Using Fonts for Emphasis. We place the key terms and the index's page reference for each defining occurrence in **bold** text for easy reference. We emphasize on-screen components in the **bold Helvetica** font (for example, the **File** menu) and Visual C# program text in the Lucida font (for example, `int count = 5`).

Objectives. The opening quotes are followed by a list of chapter objectives.

Illustrations/Figures. Abundant tables, line drawings, UML diagrams, programs and program outputs are included.

Programming Tips. We include programming tips to help you focus on important aspects of program development. These tips and practices represent the best we've gleaned from a combined seven decades of programming and teaching experience.



Good Programming Practice

The Good Programming Practices call attention to techniques that will help you produce programs that are clearer, more understandable and more maintainable.



Common Programming Error

Pointing out these Common Programming Errors reduces the likelihood that you'll make them.



Error-Prevention Tip

These tips contain suggestions for exposing and removing bugs from your programs; many of the tips describe aspects of Visual C# that prevent bugs from getting into programs.



Performance Tip

These tips highlight opportunities for making your programs run faster or minimizing the amount of memory that they occupy.



Portability Tip

The Portability Tips help you write code that will run on a variety of platforms.



Software Engineering Observation

The Software Engineering Observations highlight architectural and design issues that affect the construction of software systems, especially large-scale systems.



Look-and-Feel Observation

These observations help you design attractive, user-friendly graphical user interfaces that conform to industry norms.

Summary Bullets. We present a section-by-section, bullet-list summary of each chapter.

Terminology. We include an alphabetized list of the important terms defined in each chapter.

Self-Review Exercises and Answers. Extensive self-review exercises *and* answers are included for self-study.

Exercises. Each chapter concludes with additional exercises including:

- simple recall of important terminology and concepts
- What’s wrong with this code?
- What does this code do?
- writing individual statements and small portions of methods and classes
- writing complete methods, classes and programs
- major projects.

Please do not write to us requesting access to the Pearson Instructor’s Resource Center which contains the book’s instructor supplements, including the exercise solutions. Access is limited strictly to college instructors teaching from the book. Instructors may obtain access only through their Pearson representatives. Solutions are *not* provided for “project” exercises. Check out our Programming Projects Resource Center for lots of additional exercise and project possibilities (www.deitel.com/ProgrammingProjects/).

Index. We’ve included an extensive index for reference. Defining occurrences of key terms are highlighted with a **bold** page number.

Student Resources and Software

This book includes the Microsoft® Visual Studio® 2010 Express Editions DVD, which contains the Visual C#® 2010 Express Edition (and other Microsoft development tools). These tools are also downloadable from

www.microsoft.com/express/Windows

We wrote *Visual C# 2010 How to Program* using Visual C#® Express Edition. You can learn more about Visual C#® at msdn.microsoft.com/vcsharp.

Deitel Online Resource Centers

Our website www.deitel.com provides Resource Centers on various topics of interest to our readers (www.deitel.com/ResourceCenters.html). We’ve found many exceptional resources online, including tutorials, documentation, software downloads, articles, blogs, podcasts, videos, code samples, books, e-books and more—most are free. Some of the Resource Centers you might find helpful while studying this book are Visual C#, ASP.NET, ASP.NET AJAX, LINQ, .NET, Silverlight, SQL Server, Web Services, Windows Communication Foundation, Windows Presentation Foundation, Windows 7, UML, Code Search Engines and Code Sites, Game Programming and Programming Projects.

Instructor Supplements

The following supplements are available to qualified instructors only through Pearson Education’s Instructor Resource Center (www.pearsonhighered.com/irc):

- *Solutions Manual* with solutions to most of the end-of-chapter exercises.
- *Test Item File* of multiple-choice questions (approximately two per book section)
- *PowerPoint® slides* containing all the code and figures in the text, plus bulleted items that summarize key points.

If you're not a registered faculty member, contact your Pearson representative or visit www.pearsonhighered.com/educator/replocator/.

CourseSmart Web Books

Today's students and instructors have increasing demands on their time and money. Pearson has responded to that need by offering digital texts and course materials online through CourseSmart. CourseSmart allows faculty to review course materials online, saving time and costs. It is also environmentally sound and offers students a high-quality digital version of the text for as much as 50% off the cost of a print copy of the text. Students receive the same content offered in the print textbook enhanced by search, note-taking, and printing tools. For more information, visit www.coursesmart.com.

Microsoft Developer Network Academic Alliance (MSDNAA) and Microsoft DreamSpark

Microsoft Developer Network Academic Alliance (MSDNAA)—Free Microsoft Software for Academic and Research Purposes

The MSDNAA provides free software for academic and research purposes. For software direct to faculty, visit www.microsoft.com/faculty. For software for your department, visit www.msdnAA.com.

Microsoft DreamSpark—Professional Developer and Designer Tools for Students

Microsoft provides many of its developer tools to students for free via a program called DreamSpark (www.dreamspark.com). See the website for details on verifying your student status so you take advantage of this program.

Acknowledgments

We'd like to thank Abbey Deitel and Barbara Deitel of Deitel & Associates, Inc., who devoted long hours of research and writing to this book.

We are fortunate to have worked on this project with the dedicated team of publishing professionals at Pearson. We appreciate the guidance, savvy and energy of Michael Hirsch, Editor-in-Chief of Computer Science. Carole Snyder recruited the book's reviewers and managed the review process. Kristine Carney designed the book's cover. Bob Engelhardt and Jeffrey Holcomb managed the book's production.

Reviewers

We wish to acknowledge the efforts of our fourth edition reviewers. Adhering to a tight schedule, they scrutinized the text and the programs and provided countless suggestions for improving the presentation:

- Octavio Hernandez, Microsoft C# MVP, Advanced Bionics
- José Antonio González Seco, Parliament of Andalusia, Spain
- Zijiang Yang, Western Michigan University

Previous Edition Reviewers

Huanhui Hu (Microsoft Corporation), Narges Kasiri (Oklahoma State University), Charles Liu (University of Texas at San Antonio), Dr. Hamid R. Nemat (The University

of North Carolina at Greensboro), Jeffrey P. Scott (Blackhawk Technical College), José Antonio González Seco (Parliament of Andalusia, Spain), Douglas B. Bock (MCSD.NET, Southern Illinois University Edwardsville), Dan Crevier (Microsoft), Amit K. Ghosh (University of Texas at El Paso), Marcelo Guerra Hahn (Microsoft), Kim Hamilton (Software Design Engineer at Microsoft and co-author of *Learning UML 2.0*), James Edward Keyser (Florida Institute of Technology), Helena Kotas (Microsoft), Chris Lovett (Software Architect at Microsoft), Bashar Lulu (INETA Country Leader, Arabian Gulf), John McIlhinney (Spatial Intelligence; Microsoft MVP 2008 Visual Developer, Visual Basic), Ged Mead (Microsoft Visual Basic MVP, DevCity.net), Anand Mukundan (Architect, Polaris Software Lab Ltd.), Timothy Ng (Microsoft), Akira Onishi (Microsoft), Joe Stagner (Senior Program Manager, Developer Tools & Platforms), Erick Thompson (Microsoft) and Jesús Ubaldo Quevedo-Torrero (University of Wisconsin–Parkside, Department of Computer Science)

Well, there you have it! Visual C# 2010 is a powerful programming language that will help you write programs quickly and effectively. It scales nicely into the realm of enterprise-systems development to help organizations build their business-critical and mission-critical information systems. As you read the book, we'd appreciate your comments, criticisms, corrections and suggestions for improvement. Please address all correspondence to:

`deitel@deitel.com`

We'll respond promptly. We hope you enjoy working with *Visual C# 2010 How to Program, 4/e* as much as we enjoyed writing it!

Paul Deitel and Harvey Deitel

August 2010 Maynard, Massachusetts

About the Authors

Paul J. Deitel, CEO and Chief Technical Officer of Deitel & Associates, Inc., is a graduate of MIT, where he studied Information Technology. Through Deitel & Associates, Inc., he has delivered hundreds of C#, Visual Basic, Java, C++, C and Internet programming courses to industry clients, including Cisco, IBM, Sun Microsystems, Dell, Lucent Technologies, Fidelity, NASA at the Kennedy Space Center, the National Severe Storm Laboratory, White Sands Missile Range, Rogue Wave Software, Boeing, SunGard Higher Education, Stratus, Cambridge Technology Partners, One Wave, Hyperion Software, Adra Systems, Entergy, CableData Systems, Nortel Networks, Puma, iRobot, Invensys and many more. He and his co-author, Dr. Harvey M. Deitel, are the world's best-selling programming-language textbook authors.

Dr. Harvey M. Deitel, Chairman and Chief Strategy Officer of Deitel & Associates, Inc., has 49 years of experience in the computer field. Dr. Deitel earned B.S. and M.S. degrees from MIT and a Ph.D. from Boston University. He has extensive college teaching experience, including earning tenure and serving as the Chairman of the Computer Science Department at Boston College before founding Deitel & Associates, Inc., with his son, Paul J. Deitel. He and Paul are the co-authors of dozens of books and LiveLessons multimedia packages and they are writing many more. With translations published in Japanese, German, Russian, Chinese, Spanish, Korean, French, Polish, Italian, Portuguese, Greek, Urdu and Turkish, the Deitels' texts have earned international recognition. Dr.

Deitel has delivered hundreds of professional programming seminars to major corporations, academic institutions, government organizations and the military.

About Deitel & Associates, Inc.

Deitel & Associates, Inc., is an internationally recognized corporate training and authoring organization specializing in computer programming languages, Internet and web software technology, object-technology education and Android™ and iPhone® app development. The company provides instructor-led courses delivered at client sites worldwide on major programming languages and platforms, such as C++, Visual C++®, C, Java™, Visual C#®, Visual Basic®, XML®, Python®, object technology, Internet and web programming, Android and iPhone app development, and a growing list of additional programming and software-development courses. The founders of Deitel & Associates, Inc., are Paul J. Deitel and Dr. Harvey M. Deitel. The company's clients include many of the world's largest companies, government agencies, branches of the military, and academic institutions. Through its 34-year publishing partnership with Prentice Hall/Pearson, Deitel & Associates, Inc., publishes leading-edge programming textbooks, professional books, interactive multimedia *Cyber Classrooms*, and *LiveLessons* DVD-based and web-based video courses. Deitel & Associates, Inc., and the authors can be reached via e-mail at:

deitel@deitel.com

To learn more about Deitel & Associates, Inc., its publications and its *Dive Into*® Series Corporate Training curriculum delivered at client locations worldwide, visit:

www.deitel.com/training/

and subscribe to the free *Deitel*® *Buzz Online* e-mail newsletter at:

www.deitel.com/newsletter/subscribe.html

Individuals wishing to purchase Deitel books, and *LiveLessons* DVD and web-based training courses can do so through www.deitel.com. Bulk orders by corporations, the government, the military and academic institutions should be placed directly with Pearson. For more information, visit

www.prenhall.com/mishtml/support.html#order



Contents

Chapters 24–31 and Appendices D–G are PDF documents posted online at the book’s Companion Website (located at www.pearsonhighered.com/deitel/).

Preface **xvii**

Before You Begin **xxvii**

I Introduction to Computers, the Internet and Visual C# **I**

1.1	Introduction	2
1.2	Computer Organization	2
1.3	Personal Computing, Distributed Computing and Client/Server Computing	4
1.4	Hardware Trends	4
1.5	Microsoft’s Windows® Operating System	4
1.6	Machine Languages, Assembly Languages and High-Level Languages	5
1.7	Visual Basic	6
1.8	C, C++, Objective-C and Java	6
1.9	C#	7
1.10	The Internet and the World Wide Web	7
1.11	Extensible Markup Language (XML)	8
1.12	Introduction to Microsoft .NET	9
1.13	The .NET Framework and the Common Language Runtime	9
1.14	Test-Driving the Advanced Painter Application	10
1.15	Introduction to Object Technology	12
1.16	Wrap-Up	15
1.17	Web Resources	15

2 Dive Into® Visual C# 2010 Express **24**

2.1	Introduction	25
2.2	Overview of the Visual Studio 2010 IDE	25
2.3	Menu Bar and Toolbar	30
2.4	Navigating the Visual Studio IDE	32
2.4.1	Solution Explorer	34
2.4.2	Toolbox	35
2.4.3	Properties Window	36
2.5	Using Help	37

2.6	Using Visual Programming to Create a Simple Program that Displays Text and an Image	40
2.7	Wrap-Up	51
2.8	Web Resources	52

3 Introduction to C# Applications 60

3.1	Introduction	61
3.2	A Simple C# Application: Displaying a Line of Text	61
3.3	Creating a Simple Application in Visual C# Express	66
3.4	Modifying Your Simple C# Application	74
3.5	Formatting Text with <code>Console.Write</code> and <code>Console.WriteLine</code>	76
3.6	Another C# Application: Adding Integers	77
3.7	Memory Concepts	81
3.8	Arithmetic	82
3.9	Decision Making: Equality and Relational Operators	85
3.10	Wrap-Up	90

4 Introduction to Classes, Objects, Methods and strings 101

4.1	Introduction	102
4.2	Classes, Objects, Methods, Properties and Instance Variables	102
4.3	Declaring a Class with a Method and Instantiating an Object of a Class	103
4.4	Declaring a Method with a Parameter	107
4.5	Instance Variables and Properties	111
4.6	UML Class Diagram with a Property	115
4.7	Software Engineering with Properties and <code>set</code> and <code>get</code> Accessors	116
4.8	Auto-Implemented Properties	117
4.9	Value Types vs. Reference Types	118
4.10	Initializing Objects with Constructors	119
4.11	Floating-Point Numbers and Type <code>decimal</code>	122
4.12	Wrap-Up	128

5 Control Statements: Part I 136

5.1	Introduction	137
5.2	Algorithms	137
5.3	Pseudocode	138
5.4	Control Structures	138
5.5	<code>if</code> Single-Selection Statement	140
5.6	<code>if...else</code> Double-Selection Statement	141
5.7	<code>while</code> Repetition Statement	146
5.8	Formulating Algorithms: Counter-Controlled Repetition	147
5.9	Formulating Algorithms: Sentinel-Controlled Repetition	152
5.10	Formulating Algorithms: Nested Control Statements	160
5.11	Compound Assignment Operators	165

5.12	Increment and Decrement Operators	165
5.13	Simple Types	168
5.14	Wrap-Up	169

6 Control Statements: Part 2 **183**

6.1	Introduction	184
6.2	Essentials of Counter-Controlled Repetition	184
6.3	for Repetition Statement	186
6.4	Examples Using the for Statement	190
6.5	do...while Repetition Statement	194
6.6	switch Multiple-Selection Statement	196
6.7	break and continue Statements	203
6.8	Logical Operators	205
6.9	Structured-Programming Summary	211
6.10	Wrap-Up	216

7 Methods: A Deeper Look **226**

7.1	Introduction	227
7.2	Packaging Code in C#	227
7.3	static Methods, static Variables and Class Math	229
7.4	Declaring Methods with Multiple Parameters	232
7.5	Notes on Declaring and Using Methods	236
7.6	Method-Call Stack and Activation Records	237
7.7	Argument Promotion and Casting	237
7.8	The .NET Framework Class Library	239
7.9	Case Study: Random-Number Generation	241
7.9.1	Scaling and Shifting Random Numbers	245
7.9.2	Random-Number Repeatability for Testing and Debugging	245
7.10	Case Study: A Game of Chance (Introducing Enumerations)	246
7.11	Scope of Declarations	251
7.12	Method Overloading	253
7.13	Optional Parameters	256
7.14	Named Parameters	257
7.15	Recursion	258
7.16	Passing Arguments: Pass-by-Value vs. Pass-by-Reference	261
7.17	Wrap-Up	264

8 Arrays **280**

8.1	Introduction	281
8.2	Arrays	281
8.3	Declaring and Creating Arrays	282
8.4	Examples Using Arrays	284
8.5	Case Study: Card Shuffling and Dealing Simulation	293
8.6	foreach Statement	298

x Contents

8.7	Passing Arrays and Array Elements to Methods	299
8.8	Passing Arrays by Value and by Reference	301
8.9	Case Study: Class <code>GradeBook</code> Using an Array to Store Grades	305
8.10	Multidimensional Arrays	310
8.11	Case Study: Class <code>GradeBook</code> Using a Rectangular Array	315
8.12	Variable-Length Argument Lists	321
8.13	Using Command-Line Arguments	322
8.14	Wrap-Up	324

9 Introduction to LINQ and the `List` Collection 344

9.1	Introduction	345
9.2	Querying an Array of <code>int</code> Values Using LINQ	346
9.3	Querying an Array of <code>Employee</code> Objects Using LINQ	350
9.4	Introduction to Collections	355
9.5	Querying a Generic Collection Using LINQ	358
9.6	Wrap-Up	360
9.7	Deitel LINQ Resource Center	360

10 Classes and Objects: A Deeper Look 364

10.1	Introduction	365
10.2	Time Class Case Study	365
10.3	Controlling Access to Members	369
10.4	Referring to the Current Object's Members with the <code>this</code> Reference	370
10.5	Time Class Case Study: Overloaded Constructors	372
10.6	Default and Parameterless Constructors	378
10.7	Composition	379
10.8	Garbage Collection and Destructors	382
10.9	<code>static</code> Class Members	383
10.10	<code>readonly</code> Instance Variables	386
10.11	Data Abstraction and Encapsulation	388
10.12	Class View and Object Browser	389
10.13	Object Initializers	391
10.14	Wrap-Up	391

11 Object-Oriented Programming: Inheritance 398

11.1	Introduction	399
11.2	Base Classes and Derived Classes	400
11.3	<code>protected</code> Members	402
11.4	Relationship between Base Classes and Derived Classes	403
11.4.1	Creating and Using a <code>CommissionEmployee</code> Class	403
11.4.2	Creating a <code>BasePlusCommissionEmployee</code> Class without Using Inheritance	408
11.4.3	Creating a <code>CommissionEmployee</code> – <code>BasePlusCommissionEmployee</code> Inheritance Hierarchy	414

11.4.4	CommissionEmployee–BasePlusCommissionEmployee Inheritance Hierarchy Using protected Instance Variables	417
11.4.5	CommissionEmployee–BasePlusCommissionEmployee Inheritance Hierarchy Using private Instance Variables	421
11.5	Constructors in Derived Classes	426
11.6	Software Engineering with Inheritance	427
11.7	Class object	428
11.8	Wrap-Up	429

12 OOP: Polymorphism, Interfaces and Operator Overloading **435**

12.1	Introduction	436
12.2	Polymorphism Examples	438
12.3	Demonstrating Polymorphic Behavior	439
12.4	Abstract Classes and Methods	442
12.5	Case Study: Payroll System Using Polymorphism	444
12.5.1	Creating Abstract Base Class Employee	445
12.5.2	Creating Concrete Derived Class SalariedEmployee	448
12.5.3	Creating Concrete Derived Class HourlyEmployee	449
12.5.4	Creating Concrete Derived Class CommissionEmployee	451
12.5.5	Creating Indirect Concrete Derived Class BasePlusCommissionEmployee	452
12.5.6	Polymorphic Processing, Operator is and Downcasting	454
12.5.7	Summary of the Allowed Assignments Between Base-Class and Derived-Class Variables	459
12.6	sealed Methods and Classes	460
12.7	Case Study: Creating and Using Interfaces	460
12.7.1	Developing an IPayable Hierarchy	462
12.7.2	Declaring Interface IPayable	463
12.7.3	Creating Class Invoice	463
12.7.4	Modifying Class Employee to Implement Interface IPayable	465
12.7.5	Modifying Class SalariedEmployee for Use with IPayable	466
12.7.6	Using Interface IPayable to Process Invoices and Employees Polymorphically	468
12.7.7	Common Interfaces of the .NET Framework Class Library	470
12.8	Operator Overloading	471
12.9	Wrap-Up	474

13 Exception Handling: A Deeper Look **479**

13.1	Introduction	480
13.2	Example: Divide by Zero without Exception Handling	480
13.3	Example: Handling DivideByZeroExceptions and FormatExceptions	483
13.3.1	Enclosing Code in a try Block	485
13.3.2	Catching Exceptions	486

13.3.3	Uncaught Exceptions	486
13.3.4	Termination Model of Exception Handling	487
13.3.5	Flow of Control When Exceptions Occur	488
13.4	.NET Exception Hierarchy	488
13.4.1	Class <code>SystemException</code>	488
13.4.2	Determining Which Exceptions a Method Throws	489
13.5	<code>finally</code> Block	490
13.6	The <code>using</code> Statement	496
13.7	Exception Properties	497
13.8	User-Defined Exception Classes	502
13.9	Wrap-Up	505

14 Graphical User Interfaces with Windows Forms: Part 1 **510**

14.1	Introduction	511
14.2	Windows Forms	512
14.3	Event Handling	514
14.3.1	A Simple Event-Driven GUI	514
14.3.2	Visual Studio Generated GUI Code	516
14.3.3	Delegates and the Event-Handling Mechanism	518
14.3.4	Another Way to Create Event Handlers	519
14.3.5	Locating Event Information	519
14.4	Control Properties and Layout	521
14.5	Labels, TextBoxes and Buttons	525
14.6	GroupBoxes and Panels	528
14.7	CheckBoxes and RadioButtons	531
14.8	PictureBoxes	539
14.9	ToolTips	541
14.10	NumericUpDown Control	543
14.11	Mouse-Event Handling	545
14.12	Keyboard-Event Handling	548
14.13	Wrap-Up	551

15 Graphical User Interfaces with Windows Forms: Part 2 **561**

15.1	Introduction	562
15.2	Menus	562
15.3	MonthCalendar Control	571
15.4	DateTimePicker Control	572
15.5	LinkLabel Control	575
15.6	ListBox Control	579
15.7	CheckedListBox Control	583
15.8	ComboBox Control	586
15.9	TreeView Control	590

15.10	ListView Control	595
15.11	TabControl Control	601
15.12	Multiple Document Interface (MDI) Windows	606
15.13	Visual Inheritance	613
15.14	User-Defined Controls	618
15.15	Wrap-Up	622

16 Strings and Characters 630

16.1	Introduction	631
16.2	Fundamentals of Characters and Strings	632
16.3	string Constructors	633
16.4	string Indexer, Length Property and CopyTo Method	634
16.5	Comparing strings	635
16.6	Locating Characters and Substrings in strings	638
16.7	Extracting Substrings from strings	641
16.8	Concatenating strings	642
16.9	Miscellaneous string Methods	643
16.10	Class StringBuilder	644
16.11	Length and Capacity Properties, EnsureCapacity Method and Indexer of Class StringBuilder	645
16.12	Append and AppendFormat Methods of Class StringBuilder	647
16.13	Insert, Remove and Replace Methods of Class StringBuilder	649
16.14	Char Methods	652
16.15	(Online) Introduction to Regular Expressions	654
16.16	Wrap-Up	655

17 Files and Streams 661

17.1	Introduction	662
17.2	Data Hierarchy	662
17.3	Files and Streams	664
17.4	Classes File and Directory	665
17.5	Creating a Sequential-Access Text File	674
17.6	Reading Data from a Sequential-Access Text File	683
17.7	Case Study: Credit Inquiry Program	687
17.8	Serialization	693
17.9	Creating a Sequential-Access File Using Object Serialization	694
17.10	Reading and Deserializing Data from a Binary File	698
17.11	Wrap-Up	700

18 Databases and LINQ 707

18.1	Introduction	708
18.2	Relational Databases	709
18.3	A Books Database	710
18.4	LINQ to SQL	713

18.5	Querying a Database with LINQ	714
18.5.1	Creating LINQ to SQL Classes	715
18.5.2	Data Bindings Between Controls and the LINQ to SQL Classes	718
18.6	Dynamically Binding Query Results	722
18.6.1	Creating the Display Query Results GUI	723
18.6.2	Coding the Display Query Results Application	723
18.7	Retrieving Data from Multiple Tables with LINQ	725
18.8	Creating a Master/Detail View Application	731
18.8.1	Creating the Master/Detail GUI	732
18.8.2	Coding the Master/Detail Application	733
18.9	Address Book Case Study	736
18.9.1	Creating the Address Book Application's GUI	738
18.9.2	Coding the Address Book Application	739
18.10	Tools and Web Resources	741
18.11	Wrap-Up	742

19 Web App Development with ASP.NET 748

19.1	Introduction	749
19.2	Web Basics	750
19.3	Multitier Application Architecture	751
19.4	Your First Web Application	753
19.4.1	Building the WebTime Application	755
19.4.2	Examining WebTime.aspx 's Code-Behind File	764
19.5	Standard Web Controls: Designing a Form	764
19.6	Validation Controls	769
19.7	Session Tracking	775
19.7.1	Cookies	776
19.7.2	Session Tracking with HttpSessionState	777
19.7.3	Options.aspx : Selecting a Programming Language	780
19.7.4	Recommendations.aspx : Displaying Recommendations Based on Session Values	783
19.8	Case Study: Database-Driven ASP.NET Guestbook	785
19.8.1	Building a Web Form that Displays Data from a Database	787
19.8.2	Modifying the Code-Behind File for the Guestbook Application	790
19.9	Online Case Study: ASP.NET AJAX	792
19.10	Online Case Study: Password-Protected Books Database Application	792
19.11	Wrap-Up	792

20 Searching and Sorting 799

20.1	Introduction	800
20.2	Searching Algorithms	801
20.2.1	Linear Search	801
20.2.2	Binary Search	805
20.3	Sorting Algorithms	810
20.3.1	Selection Sort	810

20.3.2	Insertion Sort	814
20.3.3	Merge Sort	818
20.4	Summary of the Efficiency of Searching and Sorting Algorithms	824
20.5	Wrap-Up	824

21 Data Structures **830**

21.1	Introduction	831
21.2	Simple-Type structs, Boxing and Unboxing	831
21.3	Self-Referential Classes	832
21.4	Linked Lists	833
21.5	Stacks	846
21.6	Queues	850
21.7	Trees	853
21.7.1	Binary Search Tree of Integer Values	854
21.7.2	Binary Search Tree of IComparable Objects	861
21.8	Wrap-Up	866

22 Generics **873**

22.1	Introduction	874
22.2	Motivation for Generic Methods	875
22.3	Generic-Method Implementation	877
22.4	Type Constraints	880
22.5	Overloading Generic Methods	882
22.6	Generic Classes	883
22.7	Wrap-Up	892

23 Collections **898**

23.1	Introduction	899
23.2	Collections Overview	899
23.3	Class Array and Enumerators	902
23.4	Nongeneric Collections	905
23.4.1	Class ArrayList	905
23.4.2	Class Stack	909
23.4.3	Class Hashtable	912
23.5	Generic Collections	917
23.5.1	Generic Class SortedDictionary	917
23.5.2	Generic Class LinkedList	919
23.6	Covariance and Contravariance for Generic Types	923
23.7	Wrap-Up	925

Chapters on the Web **932**

A	Operator Precedence Chart	933
B	Simple Types	935
C	ASCII Character Set	937
	Appendices on the Web	938
	Index	939

Chapters 24–31 and Appendices D–G are PDF documents posted online at the book's Companion Website (located at www.pearsonhighered.com/deitel/).

24	GUI with Windows Presentation Foundation	
25	WPF Graphics and Multimedia	
26	XML and LINQ to XML	
27	Web App Development with ASP.NET: A Deeper Look	
28	Windows Communication Foundation (WCF) Web Services	
29	Silverlight and Rich Internet Applications	
30	ATM Case Study, Part 1: Object-Oriented Design with the UML	
31	ATM Case Study, Part 2: Implementing an Object-Oriented Design	
D	Number Systems	
E	UML 2: Additional Diagram Types	
F	Unicode®	
G	Using the Visual C# 2010 Debugger	

Introduction to Computers, the Internet and Visual C#

I

The chief merit of language is clearness.

—Galen

Our life is frittered away with detail. . . . Simplify, simplify.

—Henry David Thoreau

Man is still the most extraordinary computer of all.

—John F. Kennedy

Objectives

In this chapter you'll learn:

- Basic hardware and software concepts.
- The different types of programming languages.
- The history of the Visual C# programming language.
- Some basics of object technology.
- The history of the Internet and the World Wide Web.
- About Microsoft's .NET initiative, which involves the Internet in developing and using software systems.
- To test-drive a Visual C# 2010 drawing application.

Self-Review Exercises and Answers

1.1 Fill in the blanks in each of the following statements:

- a) Computers can directly understand only their native _____ language, which is composed only of 1s and 0s.

ANS: machine.

- b) Computers process data under the control of sets of instructions called computer _____.

ANS: programs.

- c) The three types of computer programming languages discussed in the chapter are machine languages, _____ and _____.

ANS: assembly languages, high-level languages.

- d) Programs that translate high-level-language programs into machine language are called _____.

ANS: compilers.

- e) Visual Studio is a(n) _____ in which C# programs are developed.

ANS: IDE.

- f) Visual Basic is a(n) _____, event-driven language.

ANS: object-oriented.

- g) _____ is a programming language that was created by Microsoft specifically for the .NET platform.

ANS: C#.

- h) _____ is a language for sharing information via “hyperlinked” text documents on the World Wide Web.

ANS: HyperText Markup Language (HTML).

- i) The _____ executes .NET programs.

ANS: Common Language Runtime (CLR).

- j) Objects have _____ (also called properties) and perform actions (also called methods or _____).

ANS: attributes, behaviors.

- k) _____ models software in terms similar to those that people use to describe real-world objects.

ANS: Object-oriented design (OOD).

- l) With _____, you can build much of the new software you’ll need by combining existing classes.

ANS: object technology.

1.2 State whether each of the following is *true* or *false*. If *false*, explain why.

- a) The UML is used primarily to implement object-oriented systems.

ANS: False. The UML is used primarily to design object-oriented systems.

- b) C# is an object-oriented language.

ANS: True.

- c) C# is the only language available for programming .NET applications.

ANS: False. C# is one of many .NET languages (others are Visual Basic and Visual C++).

- d) Computers can directly understand high-level languages.

ANS: False. Computers can directly understand only their own machine languages.

- e) MSIL is the common intermediate format to which all .NET programs compile, regardless of their original .NET language.

ANS: True.

- f) The .NET Framework is portable to non-Windows platforms.

ANS: True.

Exercises

1.3 Categorize each of the following items as either hardware or software:

- a) CPU.
ANS: Hardware.
- b) Compiler.
ANS: Software.
- c) Input unit.
ANS: Hardware.
- d) A word-processor program.
ANS: Software.
- e) A Visual Basic program.
ANS: Software.

1.4 Translator programs, such as assemblers and compilers, convert programs from one language (referred to as the source language) to another language (referred to as the target language). Determine which of the following statements are *true* and which are *false*:

- a) An assembler translates source-language programs into machine-language programs.
ANS: True.
- b) High-level languages are generally machine dependent.
ANS: False. A high-level language must be compiled into machine-dependent language before it can be executed. This allows high-level languages to be used on all computers with appropriate compilers.
- c) A machine-language program requires translation before it can be run on a computer.
ANS: False. A machine-language program is native to a specific machine.
- d) The C# compiler translates high-level-language programs into SMIL.
ANS: False. The C# compiler translates high-level-language programs into MSIL (Microsoft Intermediate Language).

1.5 Expand each of the following acronyms:

- a) W3C.
ANS: World Wide Web Consortium.
- b) XML.
ANS: Extensible Markup Language.
- c) OOP.
ANS: Object-Oriented Programming.
- d) CLR.
ANS: Common Language Runtime.
- e) CLI.
ANS: Common Language Infrastructure.
- f) MSIL.
ANS: Microsoft Intermediate Language.
- g) UML
ANS: Unified Modeling Language.
- h) IDE
ANS: Integrated Development Environment.

1.6 What are the key benefits of the .NET Framework and the CLR? What are the drawbacks?

ANS: The key benefits are portability between operating systems and interoperability between languages. As long as a CLR exists for a platform, it can run any .NET program. Programmers can concentrate on program logic instead of platform-specific details. Thus, the double compilation (code-to-MSIL, and MSIL-to-machine code) allows for platform independence: Programs can be written once and executed on any

platform supporting the CLR—this is known as platform independence. Code written once could easily be used on another machine without modification, saving time and money. A second benefit of the .NET framework is language interoperability—software components written in different languages can interact (language independence). A drawback associated with these features is that .NET programs cannot be run until the .NET Framework is developed for a platform. Another is the overhead of the double compilation that is needed before a .NET-language program can be executed.

1.7 What are the advantages to using object-oriented techniques?

ANS: Programs that use object-oriented programming techniques are easier to understand, correct and modify. The key advantage with using object-oriented programming is that it tends to produce software that is more understandable, because it is better organized and has fewer maintenance requirements than software produced with earlier methodologies. OOP helps the programmer build applications faster by reusing existing software components that model items in the real world. OOP also helps programmers create new software components that can be reused on future software development projects. Building software quickly, correctly, and economically has been an elusive goal in the software industry. The modular, object-oriented design and implementation approach has been found to increase productivity while reducing development time, errors, and cost.

1.8 You are probably wearing on your wrist one of the world's most common types of objects—a watch. Discuss how each of the following terms and concepts applies to the notion of a watch: object, attributes and behaviors.

ANS: The entire watch is an object that is composed of many other objects (the moving parts, the band, the face, etc.) Watch attributes are time, color, band style, technology (digital or analog), and the like. The behaviors of the watch include setting the time and getting the time. A watch can be considered a specific type of clock (as can an alarm clock).

1.9 What was the key reason that Visual Basic was developed as a special version of the BASIC programming language?

ANS: To make it convenient to develop applications that would run on the Microsoft Windows operating system.

1.10 What is the key accomplishment of the UML?

ANS: It replaces the many different graphical modeling languages with a single (unified) language for modeling that can be used by developers regardless of the different OOAD processes they may use.

1.11 What did the chief benefit of the early Internet prove to be?

ANS: Communication by e-mail. Today, that communication is also facilitated by applications such as instant messaging and file transfer.

1.12 What is the key capability of the web?

ANS: It allows computer users to locate and view multimedia-based documents on almost any subject over the Internet.

1.13 What is the key vision of Microsoft's .NET initiative?

ANS: To embrace the Internet and the web in the development and use of software.

1.14 How does the .NET Framework Class Library facilitate the development of .NET applications?

ANS: First, the Framework Class Library is a large library of reusable classes that reduces development time. Programmers can build software quickly by reusing framework's classes, rather than building new classes "from scratch." Second, the Framework Class Library is shared by all of the .NET languages, which means that programmers who work in multiple languages have to learn only one class library.

1.15 What is the key advantage of standardizing .NET's CLI (Common Language Infrastructure)?

ANS: Standardization makes it easier to create the .NET Framework for other platforms besides Microsoft Windows. This encourages the spread of .NET, enabling existing .NET applications to run on more systems of different types.

1.16 Besides the obvious benefits of reuse made possible by OOP, what do many organizations report as another key benefit of OOP?

ANS: That OOP tends to produce software that is more understandable, better organized, and easier to maintain, modify and debug.

1.17 Why is XML so crucial to the development of future software systems?

ANS: XML is likely to become the universal technology for data representation.

Making a Difference Exercises

1.18 (*Test Drive: Carbon Footprint Calculator*) Some scientists believe that carbon emissions, especially from the burning of fossil fuels, contribute significantly to global warming and that this can be combatted if individuals take steps to limit their use of carbon-based fuels. Organizations and individuals are increasingly concerned about their "carbon footprints." Websites such as TerraPass

www.terrapass.com/carbon-footprint-calculator/

and Carbon Footprint

www.carbonfootprint.com/calculator.aspx

provide carbon footprint calculators. Test drive these calculators to determine your carbon footprint. Exercises in later chapters will ask you to program your own carbon footprint calculator. To prepare for this, use the web to research the formulas for calculating carbon footprints.

1.19 (*Test Drive: Body Mass Index Calculator*) By recent estimates, two-thirds of the people in the United States are overweight and about half of those are obese. This causes significant increases in illnesses such as diabetes and heart disease. To determine whether a person is overweight or obese, you can use a measure called the body mass index (BMI). The United States Department of Health and Human Services provides a BMI calculator at www.nhlbhsupport.com/bmi/. Use it to calculate your own BMI. A forthcoming exercise will ask you to program your own BMI calculator. To prepare for this, use the web to research the formulas for calculating BMI.

1.20 (*Attributes of Hybrid Vehicles*) In this chapter you learned some basics of classes. Now you'll "flesh out" aspects of a class called "Hybrid Vehicle." Hybrid vehicles are becoming increasingly popular, because they often get much better mileage than purely gasoline-powered vehicles. Browse the web and study the features of four or five of today's popular hybrid cars, then list as many of their hybrid-related attributes as you can. Some common attributes include city-miles-per-gallon and highway-miles-per-gallon. Also list the attributes of the batteries (type, weight, etc.).

ANS:

- Manufacturer
- Type of Hybrid—Battery hybrid (Hybrid Electric Vehicles), Plug-in hybrid, Fuel cell etc.

- Driver feedback system—so the driver can monitor fuel efficiency based on their driving
- Energy recovery—for example, regenerative braking
- Carbon footprint—tons of CO₂ per year
- Fuel capacity
- City-miles-per-gallon
- Highway-miles-per-gallon
- Two-mode hybrid propulsion system
- Engine size—V6, V8, etc.
- Vehicle type—SUV, crossover, compact, mid-size, etc.
- Seating capacity
- Horse power
- Drive train (front wheel drive, all wheel drive)
- Top speed
- Torque
- Price

1.21 (*Gender Neutrality*) Many people want to eliminate sexism in all forms of communication. You've been asked to create a program that can process a paragraph of text and replace gender-specific words with gender-neutral ones. Assuming that you've been given a list of gender-specific words and their gender-neutral replacements (e.g., replace "wife" with "spouse," "man" with "person," "daughter" with "child" and so on), explain the procedure you'd use to read through a paragraph of text and manually perform these replacements. How might your procedure generate a strange term like "woperchild?" In Chapter 5, you'll learn that a more formal term for "procedure" is "algorithm," and that an algorithm specifies the steps to be performed and the order in which to perform them.

ANS: Search through the entire paragraph for a word such as "wife" and replace every occurrence with "spouse." Repeat this searching process for every gender specific word in the list. You could accidentally get a word like "woperchild" if you are not careful about how you perform replacements. For example, the word "man" can be part of a larger word, like "woman." So, replacing every occurrence of "man" can yield strange results. Consider the process of replacing "man" with "person" then replacing "son" with "child." If you encounter the word "woman," which contains the word "man," you'd replace "man" with "person" resulting in the word "woperson." In a subsequent pass you'd encounter "woperson" and replace "son" with "child" resulting in the "woperchild."

Dive Into® Visual C# 2010 Express

2

Seeing is believing.

—Proverb

Form ever follows function.

—Louis Henri Sullivan

*Intelligence ... is the faculty of
making artificial objects,
especially tools to make tools.*

—Henri-Louis Bergson

Objectives

In this chapter you'll learn:

- The basics of the Visual Studio Integrated Development Environment (IDE) that assists you in writing, running and debugging your Visual Basic programs.
- Visual Studio's help features.
- Key commands contained in the IDE's menus and toolbars.
- The purpose of the various kinds of windows in the Visual Studio 2008 IDE.
- What visual programming is and how it simplifies and speeds program development.
- To create, compile and execute a simple Visual Basic program that displays text and an image using the Visual Studio IDE and the technique of visual programming.

Self-Review Exercises

2.1 Fill in the blanks in each of the following statements:

- a) The technique of _____ allows you to create GUIs without writing any code.
ANS: visual programming.
- b) A(n) _____ is a group of one or more projects that collectively form a Visual Basic program.
ANS: solution.
- c) The _____ feature hides a window when the mouse pointer is moved outside the window's area.
ANS: auto-hide.
- d) A(n) _____ appears when the mouse pointer hovers over an icon.
ANS: tool tip.
- e) The _____ window allows you to browse solution files.
ANS: **Solution Explorer**
- f) The properties in the **Properties** window can be sorted _____ or _____.
ANS: alphabetically, categorically.
- g) A Form's _____ property specifies the text displayed in the Form's title bar.
ANS: Text.
- h) The _____ allows you to add controls to the Form in a visual manner.
ANS: **Toolbox**.
- i) Using _____ displays relevant help articles, based on the current context.
ANS: context-sensitive help.
- j) The _____ property specifies how text is aligned within a Label's boundaries.
ANS: TextAlign.

2.2 State whether each of the following is *true* or *false*. If *false*, explain why.

- a) The title bar of the IDE displays the IDE's mode.
ANS: True.
- b) The X box toggles auto-hide.
ANS: False. The pin icon toggles auto-hide. The X box closes a window.
- c) The toolbar icons represent various menu commands.
ANS: True.
- d) The toolbar contains icons that represent controls you can drag onto a Form.
ANS: False. The **Toolbox** contains icons that represent such controls.
- e) Both Forms and Labels have a title bar.
ANS: False. Forms have a title bar but Labels do not (although they do have Label text).
- f) Control properties can be modified only by writing code.
ANS: False. Control properties can be set using the **Properties** window.
- g) PictureBoxes typically display images.
ANS: True.
- h) C# files use the file extension .chsharp.
ANS: False. C# files use the file extension .cs.
- i) A Form's background color is set using the BackColor property.
ANS: True.

Exercises

2.3 Fill in the blanks in each of the following statements:

- a) When an ellipsis button is clicked, a(n) _____ is displayed.
ANS: dialog. Dialogs are windows that facilitate user-computer communication.
- b) To save every file in a solution, select _____.
ANS: **File > Save All**.
- c) Using _____ help immediately displays a relevant help article.
ANS: context-sensitive.
- d) “GUI” is an acronym for _____.
ANS: graphical user interface.

2.4 State whether each of the following is *true* or *false*. If *false*, explain why.

- a) You can add a control to a Form by double clicking its control icon in the **Toolbox**.
ANS: True.
- b) The Form, Label and PictureBox have identical properties.
ANS: False. Each type of control has a different set of properties, although controls can have common properties.
- c) If your machine is connected to the Internet, you can browse the Internet from the Visual Studio IDE.
ANS: True.
- d) C# programmers usually create complex programs without writing any code.
ANS: False. C# programming usually involves a combination of writing a portion of the program code and having Visual Studio generate the remaining code.
- e) Sizing handles are visible during execution.
ANS: False. Sizing handles are present only in **Design** view when a Form or control is selected.

2.5 Some features that appear throughout Visual Studio perform similar actions in different contexts. Explain and give examples of how the ellipsis buttons, down-arrow buttons and tool tips act in this manner. Why do you think the Visual Studio IDE was designed this way?

ANS: An ellipsis button indicates that a dialog will be displayed when the button is clicked. The down-arrow button indicates that there are more options, both for toolbar icons and for items in the **Properties** window. Moving the mouse pointer over most icons displays the icon's name as a tool tip. These features make the Visual Studio IDE easier to learn and use.

2.6 Fill in the blanks in each of the following statements:

- a) The _____ property specifies which image a PictureBox displays.
ANS: Image.
- b) The _____ menu contains commands for arranging and displaying windows.
ANS: **Window**

2.7 Briefly describe each of the following IDE features:

- a) toolbar
ANS: A toolbar contains icons that, when clicked, execute a command.
- b) menu bar
ANS: A menu bar contains menus, which are groups of related commands.
- c) **Toolbox**
ANS: The **Toolbox** contains controls used to customize forms.
- d) control
ANS: A control is a component, such as a PictureBox or Label. Controls are added to a Form.
- e) Form

ANS: A Form represents the Windows Forms application that you are creating. The Form and controls collectively represent the program's GUI.

f) solution

ANS: A solution is a group of projects.

[*Note:* In the following exercises, you are asked to create GUIs using controls that we have not yet discussed in this book. The exercises give you practice with visual programming only—the programs do not perform any actions. You place controls from the **Toolbox** on a Form to familiarize yourself with what each control looks like. We have provided step-by-step instructions for you. If you follow these, you should be able to replicate the screen images we provide.]

2.8 (*Notepad GUI*) Create the GUI for the notepad as shown in Fig. 2.48.

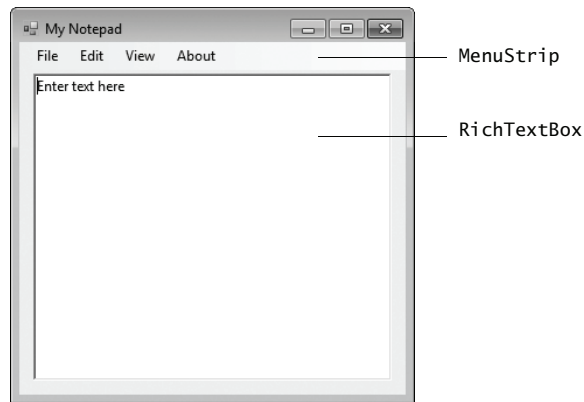


Fig. 2.48 | Notepad GUI.

- Manipulating the Form's properties.** Change the Text property of the Form to My Notepad. Change the Font property to 9pt Segoe UI.
- Adding a MenuStrip control to the Form.** Add a MenuStrip to the Form. After inserting the MenuStrip, add items by clicking the **Type Here** section, typing a menu name (e.g., **File**, **Edit**, **View** and **About**) and then pressing *Enter*.
- Adding a RichTextBox to the Form.** Drag this control onto the Form. Use the sizing handles to resize and position the RichTextBox as shown in Fig. 2.48. Change the Text property to Enter text here.

2.9 (Calendar and Appointments GUI) Create the GUI for the calendar as shown in Fig. 2.49.

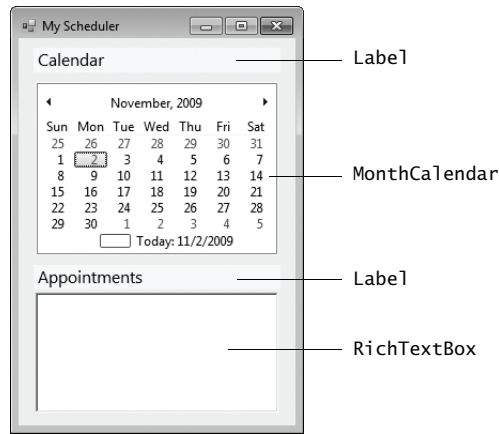


Fig. 2.49 | Calendar and appointments GUI.

- Manipulating the Form's properties.** Change the Text property of the Form to My Scheduler. Change the Font property to 9pt Segoe UI. Set the Form's Size property to 275, 400.
- Adding Labels to the Form.** Add two Labels to the Form. Both should be of equal size (231, 23; remember to set the AutoSize property to False) and should be centered in the Form horizontally, as shown. Set the Label's Text properties to match Fig. 2.49. Use 12-point font size. Also, set the BackColor property to Yellow.
- Adding a MonthCalendar control to the Form.** Add this control to the Form and center it horizontally in the appropriate place between the two Labels.
- Adding a RichTextBox control to the Form.** Add a RichTextBox control to the Form and center it below the second Label. Resize the RichTextBox accordingly.

2.10 (Calculator GUI) Create the GUI for the calculator as shown in Fig. 2.50.

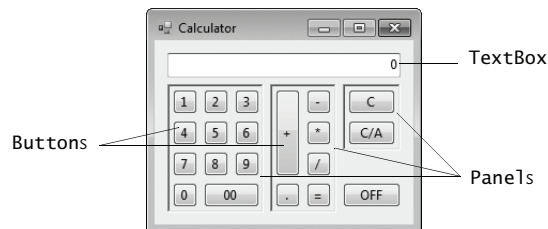


Fig. 2.50 | Calculator GUI.

- Manipulating the Form's properties.** Change the Text property of the Form to Calculator. Change the Font property to 9pt Segoe UI. Change the Size property of the Form to 258, 210.

- b) *Adding a TextBox to the Form.* Set the TextBox's Text property in the **Properties** window to 0. Stretch the TextBox and position it as shown in Fig. 2.50. Set the TextAlign property to Right—this right aligns text displayed in the TextBox.
- c) *Adding the first Panel to the Form.* Panel controls are used to group other controls. Add a Panel to the Form. Change the Panel's BorderStyle property to Fixed3D to make the inside of the Panel appear recessed. Change the Size property to 90, 120. This Panel will contain the calculator's numeric keys.
- d) *Adding the second Panel to the Form.* Change the Panel's BorderStyle property to Fixed3D. Change the Size property to 62, 120. This Panel will contain the calculator's operator keys.
- e) *Adding the third (and last) Panel to the Form.* Change the Panel's BorderStyle property to Fixed3D. Change the Size property to 54, 62. This Panel contains the calculator's C (clear) and C/A (clear all) keys.
- f) *Adding Buttons to the Form.* There are 20 Buttons on the calculator. Add a Button to the Panel by dragging and dropping it on the Panel. Change the Text property of each Button to the calculator key it represents. The value you enter in the Text property will appear on the face of the Button. Finally, resize the Buttons, using their Size properties. Each Button labeled 0–9, *, /, -, = and . should have a size of 23, 23. The 00 Button has size 52, 23. The OFF Button has size 54, 23. The + Button is sized 25, 64. The C (clear) and C/A (clear all) Buttons are sized 44, 23.

2.11 (Alarm Clock GUI) Create the GUI for the alarm clock as shown in Fig. 2.51.

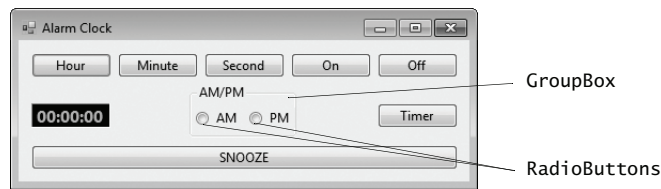


Fig. 2.51 | Alarm clock GUI.

- a) *Manipulating the Form's properties.* Change the Text property of the Form to Alarm Clock. Change the Font property to 9pt Segoe UI. Change the Size property of the Form to 438, 170.
- b) *Adding Buttons to the Form.* Add six Buttons to the Form. Change the Text property of each Button to the appropriate text. Align the Buttons as shown.
- c) *Adding a GroupBox to the Form.* GroupBoxes are like Panels, except that GroupBoxes display a title. Change the Text property to AM/PM, and set the Size property to 100, 50. Center the GroupBox horizontally on the Form.
- d) *Adding AM/PM RadioButtons to the GroupBox.* Place two RadioButtons in the GroupBox. Change the Text property of one RadioButton to AM and the other to PM. Align the RadioButtons as shown.
- e) *Adding the time Label to the Form.* Add a Label to the Form and change its Text property to 00:00:00. Change the BorderStyle property to Fixed3D and the BackColor to Black. Use the Font property to make the time bold and 12pt. Change the ForeColor to Silver (located in the **Web** tab) to make the time stand out against the black background. Position the Label as shown.

2.12 (Radio GUI) Create the GUI for the radio as shown in Fig. 2.52. [Note: The image used in this exercise is located in the examples folder for Chapter 2.]

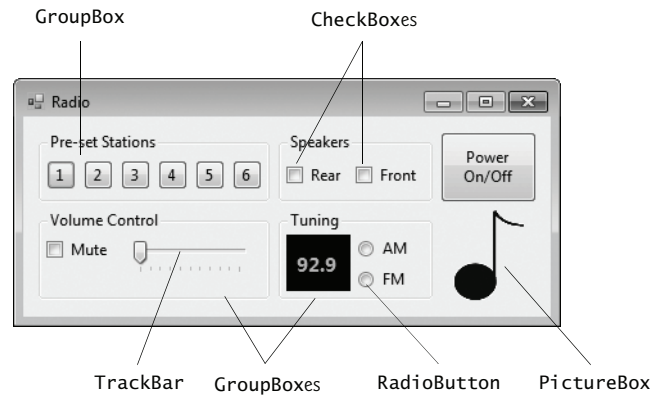



Fig. 2.52 | Radio GUI.

- a) *Manipulating the Form's properties.* Change the Font property to 9pt Segoe UI. Change the Form's Text property to Radio and the Size to 427, 194.
- b) *Adding the Pre-set Stations GroupBox and Buttons.* Set the GroupBox's Size to 180, 55 and its Text to Pre-set Stations. Add six Buttons to the GroupBox. Set each one's Size to 23, 23. Change the Buttons' Text properties to 1, 2, 3, 4, 5, 6, respectively.
- c) *Adding the Speakers GroupBox and CheckBoxes.* Set the GroupBox's Size to 120, 55 and its Text to Speakers. Add two CheckBoxes to the GroupBox. Set the Text properties for the CheckBoxes to Rear and Front.
- d) *Adding the Power On/Off Button.* Add a Button to the Form. Set its Text to Power On/Off and its Size to 75, 55.
- e) *Adding the Volume Control GroupBox, the Mute CheckBox and the Volume TrackBar.* Add a GroupBox to the Form. Set its Text to Volume Control and its Size to 180, 70. Add a CheckBox to the GroupBox. Set its Text to Mute. Add a TrackBar to the GroupBox.
- f) *Adding the Tuning GroupBox, the radio station Label and the AM/FM RadioButtons.* Add a GroupBox to the Form. Set its Text to Tuning and its Size to 120, 70. Add a Label to the GroupBox. Set its AutoSize to False, its Size to 50, 44, its BackColor to Black, its ForeColor to Silver, its font to 12pt bold and its TextAlign to MiddleCenter. Set its Text to 92.9. Place the Label as shown in the figure. Add two RadioButtons to the GroupBox. Set the Text of one to AM and of the other to FM.
- g) *Adding the image.* Add a PictureBox to the Form. Set its BackColor to PeachPuff, its SizeMode to StretchImage and its Size to 55, 70. Set the Image property to Music-Note.gif (located in the examples folder for Chapter 2).

Introduction to C# Applications

3



*What's in a name?
That which we call a rose
by any other name
would smell as sweet.*

—William Shakespeare

*When faced with a decision, I
always ask, "What would be the
most fun?"*

—Peggy Walker

Objectives

In this chapter you'll learn:

- To write simple C# applications using code rather than visual programming.
- To input data from the keyboard and output data to the screen.
- To declare and use data of various types.
- To store and retrieve data from memory.
- To use arithmetic operators.
- To determine the order in which operators are applied.
- To write decision-making statements.
- To use relational and equality operators.
- To use message dialogs to display messages.

Self-Review Exercises

3.1 Fill in the blanks in each of the following statements:

a) A(n) _____ begins the body of every method, and a(n) _____ ends the body of every method.

ANS: left brace { }, right brace }.

b) Most statements end with a(n) _____.

ANS: semicolon ;).

c) The _____ statement is used to make decisions.

ANS: if.

d) _____ begins a single-line comment.

ANS: //.

e) _____, _____ and _____ are called whitespace characters. Newline characters are also considered whitespace characters.

ANS: Blank lines, space characters, tab characters.

f) _____ are reserved for use by C#.

ANS: Keywords.

g) C# applications begin execution at method _____.

ANS: Main.

h) Methods _____ and _____ display information in the console window.

ANS: Console.WriteLine and Console.Write.

3.2 State whether each of the following is *true* or *false*. If *false*, explain why.

a) Comments cause the computer to display the text after the // on the screen when the application executes.

ANS: False. Comments do not cause any action to be performed when the application executes. They are used to document applications and improve their readability.

b) C# considers the variables `number` and `NUMBER` to be identical.

ANS: False. C# is case sensitive, so these variables are distinct.

c) The remainder operator (%) can be used only with integer operands.

ANS: False. The remainder operator can also be used with noninteger operands in C#.

d) The arithmetic operators *, /, %, + and - all have the same level of precedence.

ANS: False. The operators *, / and % are on the same level of precedence, and the operators + and - are on a lower level of precedence.

3.3 Write statements to accomplish each of the following tasks:

a) Declare variables `c`, `thisIsAVariable`, `q76354` and `number` to be of type `int`.

ANS: `int c, thisIsAVariable, q76354, number;`

or

`int c;`

`int thisIsAVariable;`

`int q76354;`

`int number;`

b) Prompt the user to enter an integer.

ANS: `Console.Write("Enter an integer: ");`

c) Input an integer and assign the result to `int` variable `value`.

ANS: `value = Convert.ToInt32(Console.ReadLine());`

d) If the variable `number` is not equal to 7, display "The variable number is not equal to 7".

ANS: `if (number != 7)`

`Console.WriteLine("The variable number is not equal to 7");`

e) Display "This is a C# application" on one line in the console window.

ANS: `Console.WriteLine("This is a C# application");`

- f) Display "This is a C# application" on two lines in the console window. The first line should end with C#. Use method `Console.WriteLine`.

ANS: `Console.WriteLine("This is a C#\napplication");`

- g) Display "This is a C# application" on two lines in the console window. The first line should end with C#. Use method `Console.WriteLine` and two format items.

ANS: `Console.WriteLine("{0}\n{1}", "This is a C#", "application");`

3.4 Identify and correct the errors in each of the following statements:

- a) `if (c < 7);`

`Console.WriteLine("c is less than 7");`

ANS: Error: Semicolon after the right parenthesis of the condition (`c < 7`) in the `if` statement.

Correction: Remove the semicolon after the right parenthesis. [*Note:* The error would cause the output statement to execute regardless of whether the condition in the `if` is true.]

- b) `if (c => 7)`

`Console.WriteLine("c is equal to or greater than 7");`

ANS: Error: The relational operator `=>` is incorrect.

Correction: Change `=>` to `>=`.

3.5 Write declarations, statements or comments that accomplish each of the following tasks:

- a) State that an application will calculate the product of three integers.

ANS: `// Calculating the product of three integers`

- b) Declare the variables `x`, `y`, `z` and `result` to be of type `int`.

ANS: `int x, y, z, result;`

or

`int x;`

`int y;`

`int z;`

`int result;`

- c) Prompt the user to enter the first integer.

ANS: `Console.Write("Enter first integer: ");`

- d) Read the first integer from the user and store it in the variable `x`.

ANS: `x = Convert.ToInt32(Console.ReadLine());`

- e) Prompt the user to enter the second integer.

ANS: `Console.Write("Enter second integer: ");`

- f) Read the second integer from the user and store it in the variable `y`.

ANS: `y = Convert.ToInt32(Console.ReadLine());`

- g) Prompt the user to enter the third integer.

ANS: `Console.Write("Enter third integer: ");`

- h) Read the third integer from the user and store it in the variable `z`.

ANS: `z = Convert.ToInt32(Console.ReadLine());`

- i) Compute the product of the three integers contained in variables `x`, `y` and `z`, and assign the result to the variable `result`.

ANS: `result = x * y * z;`

- j) Display the message "Product is" followed by the value of the variable `result`.

ANS: `Console.WriteLine("Product is {0}", result);`

4 Chapter 3 Introduction to C# Applications

3.6 Using the statements you wrote in Exercise 3.5, write a complete application that calculates and displays the product of three integers.

ANS:

```
1 // Exercise 3.6: Product.cs
2 // Calculating the product of three integers.
3 using System;
4
5 public class Product
6 {
7     public static void Main( string[] args )
8     {
9         int x; // stores first number to be entered by user
10        int y; // stores second number to be entered by user
11        int z; // stores third number to be entered by user
12        int result; // product of numbers
13
14        Console.Write( "Enter first integer: " ); // prompt for input
15        x = Convert.ToInt32( Console.ReadLine() ); // read first integer
16
17        Console.Write( "Enter second integer: " ); // prompt for input
18        y = Convert.ToInt32( Console.ReadLine() ); // read second integer
19
20        Console.Write( "Enter third integer: " ); // prompt for input
21        z = Convert.ToInt32( Console.ReadLine() ); // read third integer
22
23        result = x * y * z; // calculate the product of the numbers
24
25        Console.WriteLine( "Product is {0}", result );
26    } // end Main
27 } // end class Product
```

```
Enter first integer: 10
Enter second integer: 20
Enter third integer: 30
Product is 6000
```

Exercises

3.7 Fill in the blanks in each of the following statements:

a) _____ are used to document an application and improve its readability.

ANS: Comments.

b) A decision can be made in a C# application with a(n) _____.

ANS: if statement.

c) Calculations are normally performed by _____ statements.

ANS: assignment.

d) The arithmetic operators with the same precedence as multiplication are _____ and _____.

ANS: division (/), remainder (%)

e) When parentheses in an arithmetic expression are nested, the _____ set of parentheses is evaluated first.

ANS: innermost.

- f) A location in the computer's memory that may contain different values at various times throughout the execution of an application is called a(n) _____.

ANS: variable.

3.8 Write C# statements that accomplish each of the following tasks:

- a) Display the message "Enter an integer: ", leaving the cursor on the same line.

ANS: `Console.Write("Enter an integer: ");`

- b) Assign the product of variables b and c to variable a.

ANS: `a = b * c;`

- c) State that an application performs a simple payroll calculation (i.e., use text that helps to document an application).

ANS: `// This application performs a simple payroll calculation.`

3.9 State whether each of the following is *true* or *false*. If *false*, explain why.

- a) C# operators are evaluated from left to right.

ANS: False. Some operators (e.g., assignment, =) evaluate from right to left.

- b) The following are all valid variable names: `_under_bar_`, `m928134`, `t5`, `j7`, `her_sales`, `his_account_total`, `a`, `b`, `c`, `z` and `z2`.

ANS: True.

- c) A valid C# arithmetic expression with no parentheses is evaluated from left to right.

ANS: False. The expression is evaluated according to operator precedence and associativity.

- d) The following are all invalid variable names: `3g`, `87`, `67h2`, `h22` and `2h`.

ANS: False. Identifier `h22` is a valid variable name.

3.10 Assuming that `x = 2` and `y = 3`, what does each of the following statements display?

- a) `Console.WriteLine("x = {0}", x);`

ANS: `x = 2`

- b) `Console.WriteLine("Value of {0} + {0} is {1}", x, (x + x));`

ANS: Value of 2 + 2 is 4

- c) `Console.Write("x = ");`

ANS: `x =`

- d) `Console.WriteLine("{0} = {1}", (x + y), (y + x));`

ANS: `5 = 5`

3.11 Which of the following C# statements contain variables whose values are modified?

- a) `p = i + j + k + 7;`

- b) `Console.WriteLine("variables whose values are modified");`

- c) `Console.WriteLine("a = 5");`

- d) `value = Convert.ToInt32(Console.ReadLine());`

ANS: (a), (d).

3.12 Given that $y = ax^3 + 7$, which of the following are correct C# statements for this equation?

- a) `y = a * x * x * x + 7;`

- b) `y = a * x * x * (x + 7);`

- c) `y = (a * x) * x * (x + 7);`

- d) `y = (a * x) * x * x + 7;`

- e) `y = a * (x * x * x) + 7;`

- f) `y = a * x * (x * x + 7);`

ANS: (a), (d), (e).

3.13 (*Order of Evaluation*) State the order of evaluation of the operators in each of the following C# statements and show the value of x after each statement is performed:

- a) `x = 7 + 3 * 6 / 2 - 1;`

ANS: *, /, +, -, =; Value of x is 15.

6 Chapter 3 Introduction to C# Applications

b) $x = 2 \% 2 + 2 * 2 - 2 / 2;$

ANS: %, *, /, +, -, =; Value of x is 3.

c) $x = (3 * 9 * (3 + (9 * 3 / (3))));$

ANS: $x = (3 * 9 * (3 + (9 * 3 / (3))));$

6 4 5 3 1 2

Value of x is 324.

3.14 (Printing) Write an application that displays the numbers 1 to 4 on the same line, with each pair of adjacent numbers separated by one space. Write the application using the following techniques:

- Use one `Console.WriteLine` statement.
- Use four `Console.Write` statements.
- Use one `Console.WriteLine` statement with four format items.

ANS:

```
1 // Exercise 3.14 Solution: Printing.cs
2 // Displays the numbers 1 through 4 several ways.
3 using System;
4
5 public class Printing
6 {
7     public static void Main( string[] args )
8     {
9         Console.Write( "Part (a): " );
10
11         // one Console.WriteLine statement
12         Console.WriteLine( "1 2 3 4" );
13
14         Console.Write( "Part (b): " );
15
16         // four Console.Write statements
17         Console.Write( "1 " );
18         Console.Write( "2 " );
19         Console.Write( "3 " );
20         Console.Write( "4\n" );
21
22         Console.Write( "Part (c): " );
23
24         // one Console.WriteLine statement with four format items
25         Console.WriteLine( "{0} {1} {2} {3}", 1, 2, 3, 4 );
26     } // end Main
27 } // end class Printing
```

Part (a): 1 2 3 4

Part (b): 1 2 3 4

Part (c): 1 2 3 4

3.15 (*Arithmetic*) Write an application that asks the user to enter two integers, obtains them from the user and displays their sum, product, difference and quotient (division). Use the techniques shown in Fig. 3.18.

ANS:

```

1 // Exercise 3.15 Solution: Calculate.cs
2 // Displays the sum, product, difference and quotient of two numbers.
3 using System;
4
5 public class Calculate
6 {
7     public static void Main( string[] args )
8     {
9         int number1; // first number
10        int number2; // second number
11
12        // prompt for input and read first integer
13        Console.Write( "Enter first integer: " );
14        number1 = Convert.ToInt32( Console.ReadLine() );
15
16        // prompt for input and read second integer
17        Console.Write( "Enter second integer: " );
18        number2 = Convert.ToInt32( Console.ReadLine() );
19
20        // display results
21        Console.WriteLine( "\nSum is {0}", ( number1 + number2 ) );
22        Console.WriteLine( "Product is {0}", ( number1 * number2 ) );
23        Console.WriteLine( "Difference is {0}", ( number1 - number2 ) );
24        Console.WriteLine( "Quotient is {0}", ( number1 / number2 ) );
25    } // end Main
26 } // end class Calculate

```

```

Enter first integer: 45
Enter second integer: 5

```

```

Sum is 50
Product is 225
Difference is 40
Quotient is 9

```

3.16 (*Comparing Integers*) Write an application that asks the user to enter two integers, obtains them from the user and displays the larger number followed by the words "is larger". If the numbers are equal, display the message "These numbers are equal." Use the techniques shown in Fig. 3.26.

ANS:

```

1 // Exercise 3.16 Solution: Larger.cs
2 // Application that determines the larger of two numbers.
3 using System;
4
5 public class Larger
6 {

```

```

7 public static void Main( string[] args )
8 {
9     int number1; // first number to compare
10    int number2; // second number to compare
11
12    // prompt for input and read first number
13    Console.Write( "Enter first integer: " );
14    number1 = Convert.ToInt32( Console.ReadLine() );
15
16    // prompt for input and read second number
17    Console.Write( "Enter second integer: " );
18    number2 = Convert.ToInt32( Console.ReadLine() );
19
20    // determine whether number1 is greater than number2
21    if ( number1 > number2 )
22        Console.WriteLine( "{0} is larger", number1 );
23
24    // determine whether number1 is less than number2
25    if ( number1 < number2 )
26        Console.WriteLine( "{0} is larger", number2 );
27
28    // determine whether number1 is equal to number2
29    if ( number1 == number2 )
30        Console.WriteLine( "These numbers are equal" );
31    } // end Main
32 } // end class Larger

```

```

Enter first integer: 10
Enter second integer: 12
12 is larger

```

```

Enter first integer: 12
Enter second integer: 10
12 is larger

```

```

Enter first integer: 11
Enter second integer: 11
These numbers are equal

```

3.17 (*Arithmetic, Smallest and Largest*) Write an application that inputs three integers from the user and displays the sum, average, product, and smallest and largest of the numbers. Use the techniques from Fig. 3.26. [Note: The average calculation in this exercise should result in an integer representation of the average. So, if the sum of the values is 7, the average should be 2, not 2.3333....]

ANS:

```

1 // Exercise 3.17 Solution: Calculate2.cs
2 // Perform simple calculations on three integers.
3 using System;

```

```
4
5 public class Calculate2
6 {
7     public static void Main( string[] args )
8     {
9         int number1; // first number
10        int number2; // second number
11        int number3; // third number
12        int largest; // largest value
13        int smallest; // smallest value
14        int sum; // sum of numbers
15        int product; // product of numbers
16        int average; // average of numbers
17
18        // prompt for input and read first integer
19        Console.Write( "Enter first integer: " );
20        number1 = Convert.ToInt32( Console.ReadLine() );
21
22        // prompt for input and read second integer
23        Console.Write( "Enter second integer: " );
24        number2 = Convert.ToInt32( Console.ReadLine() );
25
26        // prompt for input and read third integer
27        Console.Write( "Enter third integer: " );
28        number3 = Convert.ToInt32( Console.ReadLine() );
29
30        // determine largest value
31        largest = number1; // assume number1 is the largest
32
33        if ( number2 > largest ) // determine whether number2 is larger
34            largest = number2;
35
36        if ( number3 > largest ) // determine whether number3 is larger
37            largest = number3;
38
39        // determine smallest value
40        smallest = number1; // assume number1 is the smallest
41
42        if ( number2 < smallest ) // determine whether number2 is smallest
43            smallest = number2;
44
45        if ( number3 < smallest ) // determine whether number3 is smallest
46            smallest = number3;
47
48        // perform calculations
49        sum = number1 + number2 + number3;
50        product = number1 * number2 * number3;
51        average = sum / 3;
52
53        // display results
54        Console.WriteLine( "\nFor the numbers {0}, {1} and {2}",
55            number1, number2, number3 );
56        Console.WriteLine( "Largest is {0}", largest );
57        Console.WriteLine( "Smallest is {0}", smallest );
```

```

58     Console.WriteLine( "Sum is {0}", sum );
59     Console.WriteLine( "Product is {0}", product );
60     Console.WriteLine( "Average is {0}", average );
61 } // end Main
62 } // end class Calculate2

```

```

Enter first integer: 10
Enter second integer: 20
Enter third integer: 30

For the numbers 10, 20 and 30
Largest is 30
Smallest is 10
Sum is 60
Product is 6000
Average is 20

```

```

Enter first integer: 200
Enter second integer: 300
Enter third integer: 100

For the numbers 200, 300 and 100
Largest is 300
Smallest is 100
Sum is 600
Product is 6000000
Average is 200

```

3.18 (*Displaying Shapes with Asterisks*) Write an application that displays a box, an oval, an arrow and a diamond using asterisks (*), as follows:

```

*****      ***      *      *
*      *  *      *      ***      *  *
*      *  *      *      *****      *  *
*      *  *      *      *      *      *
*      *  *      *      *      *      *
*      *  *      *      *      *      *
*      *  *      *      *      *      *
*      *  *      *      *      *      *
*****      ***      *      *

```

ANS:

```

1  // Exercise 3.18 Solution: Shapes.cs
2  // Application draws four shapes to the console window.
3  using System;
4
5  public class Shapes
6  {
7      public static void Main( string[] args )
8      {

```

```

9      Console.WriteLine( "*****      ***      *      *      " );
10     Console.WriteLine( "*      *      *      *      ***      *      " );
11     Console.WriteLine( "*      *      *      *      *****      *      " );
12     Console.WriteLine( "*      *      *      *      *      *      *      " );
13     Console.WriteLine( "*      *      *      *      *      *      *      " );
14     Console.WriteLine( "*      *      *      *      *      *      *      " );
15     Console.WriteLine( "*      *      *      *      *      *      *      " );
16     Console.WriteLine( "*      *      *      *      *      *      *      " );
17     Console.WriteLine( "*****      ***      *      *      " );
18 } // end Main
19 } // end class Shapes

```

```

*****      ***      *      *
*      *      *      *      ***      *      *
*      *      *      *      *****      *      *
*      *      *      *      *      *      *
*      *      *      *      *      *      *
*      *      *      *      *      *      *
*      *      *      *      *      *      *
*      *      *      *      *      *      *
*****      ***      *      *

```

3.19 What does the following code display?

```
Console.WriteLine( @"\n**\n***\n****\n*****" );
```

ANS:

```

*
**
***
****
*****

```

3.20 What does the following code display?

```

Console.WriteLine( "*" );
Console.WriteLine( "****" );
Console.WriteLine( "*****" );
Console.WriteLine( "*****" );
Console.WriteLine( "****" );

```

ANS:

```

*
***
*****
*****
****

```

12 Chapter 3 Introduction to C# Applications

3.21 What does the following code display?

```
Console.Write( "*" );  
Console.Write( "****" );  
Console.Write( "*****" );  
Console.Write( "*****" );  
Console.WriteLine( "*" );
```

ANS:

```
*****
```

3.22 What does the following code display?

```
Console.Write( "*" );  
Console.WriteLine( "****" );  
Console.WriteLine( "*****" );  
Console.Write( "*****" );  
Console.WriteLine( "*" );
```

ANS:

```
****  
*****  
*****
```

3.23 What does the following code display?

```
Console.WriteLine( "{0}\n{1}\n{2}", "*", "****", "*****" );
```

ANS:

```
*  
***  
*****
```

3.24 (*Odd or Even*) Write an application that reads an integer, then determines and displays whether it's odd or even. [*Hint:* Use the remainder operator. An even number is a multiple of 2. Any multiple of 2 leaves a remainder of 0 when divided by 2.]

ANS:

```
1 // Exercise 3.24 Solution: OddEven.cs  
2 // Application that determines whether a number is odd or even.  
3 using System;  
4  
5 public class OddEven  
6 {  
7     public static void Main( string[] args )  
8     {  
9         int number; // number  
10
```

```

11     Console.Write( "Enter integer: " ); // prompt for input
12     number = Convert.ToInt32( Console.ReadLine() ); // read number
13
14     // determine whether number is even
15     if ( number % 2 == 0 )
16         Console.WriteLine( "Number is even" );
17
18     // determine whether number is odd
19     if ( number % 2 != 0 )
20         Console.WriteLine( "Number is odd" );
21 } // end Main
22 } // end class OddEven

```

Enter integer: 17
Number is odd

Enter integer: 144
Number is even

3.25 (*Multiples*) Write an application that reads two integers, determines whether the first is a multiple of the second and displays the result. [*Hint: Use the remainder operator.*]
ANS:

```

1  // Exercise 3.25 Solution: Multiple.cs
2  // Application determines whether the first number entered is a multiple
3  // of the second number entered.
4  using System;
5
6  public class Multiple
7  {
8      public static void Main( string[] args )
9      {
10         int firstNumber; // stores first number user enters
11         int secondNumber; // stores second number user enters
12
13         // prompt for input and read first number
14         Console.Write( "Enter first number: " );
15         firstNumber = Convert.ToInt32( Console.ReadLine() );
16
17         // prompt for input and read second number
18         Console.Write( "Enter second number: " );
19         secondNumber = Convert.ToInt32( Console.ReadLine() );
20
21         // determine whether firstNumber is a multiple of secondNumber
22         if ( firstNumber % secondNumber == 0 )
23             Console.WriteLine( "{0} is a multiple of {1}",
24                                 firstNumber, secondNumber );
25

```

```

26      // determine whether firstNumber is not a multiple of secondNumber
27      if ( firstNumber % secondNumber != 0 )
28          Console.WriteLine( "{0} is not a multiple of {1}",
29                              firstNumber, secondNumber );
30      } // end Main
31 } // end class Multiple

```

```

Enter first number: 10
Enter second number: 2
10 is a multiple of 2

```

```

Enter first number: 17
Enter second number: 3
17 is not a multiple of 3

```

3.26 (*Diameter, Circumference and Area of a Circle*) Here's a peek ahead. In this chapter, you have learned about integers and the type `int`. C# can also represent floating-point numbers that contain decimal points, such as 3.14159. Write an application that inputs from the user the radius of a circle as an integer and displays the circle's diameter, circumference and area using the floating-point value 3.14159 for π . Use the techniques shown in Fig. 3.18. [Note: You may also use the pre-defined constant `Math.PI` for the value of π . This constant is more precise than the value 3.14159. Class `Math` is defined in namespace `System`]. Use the following formulas (r is the radius):

$$\begin{aligned} \text{diameter} &= 2r \\ \text{circumference} &= 2\pi r \\ \text{area} &= \pi r^2 \end{aligned}$$

Do not store the results of each calculation in a variable. Rather, specify each calculation as the value that will be output in a `Console.WriteLine` statement. Note that the values produced by the circumference and area calculations are floating-point numbers. You'll learn more about floating-point numbers in Chapter 4.

ANS:

```

1  // Exercise 3.26 Solution: Circle.cs
2  // Application that calculates area, circumference
3  // and diameter for a circle.
4  using System;
5
6  public class Circle
7  {
8      public static void Main( string[] args )
9      {
10         int radius; // radius of circle
11
12         Console.Write( "Enter radius: " ); // prompt for input
13         radius = Convert.ToInt32( Console.ReadLine() ); // read number
14
15         Console.WriteLine( "Diameter is {0}", ( 2 * radius ) );
16         Console.WriteLine( "Area is {0}", ( Math.PI * radius * radius ) );

```

```

17     Console.WriteLine( "Circumference is {0}",
18         ( 2 * Math.PI * radius ) );
19     } // end Main
20 } // end class Circle

```

```

Enter radius: 3
Diameter is 6
Area is 28.2743338823081
Circumference is 18.8495559215388

```

3.27 (*Integer Equivalent of a Character*) Here's another peek ahead. In this chapter, you have learned about integers and the type `int`. C# can also represent uppercase letters, lowercase letters and a considerable variety of special symbols. Every character has a corresponding integer representation. The set of characters a computer uses and the corresponding integer representations for those characters is called that computer's character set. You can indicate a character value in an application simply by enclosing that character in single quotes, as in `'A'`.

You can determine the integer equivalent of a character by preceding that character with `(int)`, as in

```
(int) 'A'
```

The keyword `int` in parentheses is known as a cast operator, and the entire expression is called a cast expression. (You'll learn about cast operators in Chapter 5.) The following statement outputs a character and its integer equivalent:

```

Console.WriteLine( "The character {0} has the value {1}\n",
    'A', ( ( int ) 'A' ) );

```

When the preceding statement executes, it displays the character `A` and the value `65` (from the so-called Unicode® character set) as part of the string.

Using statements similar to the one shown earlier in this exercise, write an application that displays the integer equivalents of some uppercase letters, lowercase letters, digits and special symbols. Display the integer equivalents of the following: `A B C a b c 0 1 2 $ * + /` and the blank character.

ANS:

```

1  // Exercise 3.27 Solution: Display.cs
2  // Application that displays a Unicode character
3  // and its integer equivalent.
4  using System;
5
6  public class Display
7  {
8      public static void Main( string[] args )
9      {
10         Console.WriteLine( "The character {0} has the value {1}",
11             'A', ( ( int ) 'A' ) );
12         Console.WriteLine( "The character {0} has the value {1}",
13             'B', ( ( int ) 'B' ) );
14         Console.WriteLine( "The character {0} has the value {1}",
15             'C', ( ( int ) 'C' ) );
16         Console.WriteLine( "The character {0} has the value {1}",
17             'a', ( ( int ) 'a' ) );

```

```

18 Console.WriteLine( "The character {0} has the value {1}",
19     'b', ( ( int ) 'b' ) );
20 Console.WriteLine( "The character {0} has the value {1}",
21     'c', ( ( int ) 'c' ) );
22 Console.WriteLine( "The character {0} has the value {1}",
23     '0', ( ( int ) '0' ) );
24 Console.WriteLine( "The character {0} has the value {1}",
25     '1', ( ( int ) '1' ) );
26 Console.WriteLine( "The character {0} has the value {1}",
27     '2', ( ( int ) '2' ) );
28 Console.WriteLine( "The character {0} has the value {1}",
29     '$', ( ( int ) '$' ) );
30 Console.WriteLine( "The character {0} has the value {1}",
31     '*', ( ( int ) '*' ) );
32 Console.WriteLine( "The character {0} has the value {1}",
33     '+', ( ( int ) '+' ) );
34 Console.WriteLine( "The character {0} has the value {1}",
35     '/', ( ( int ) '/' ) );
36 Console.WriteLine( "The character {0} has the value {1}",
37     ' ', ( ( int ) ' ' ) );
38     } // end Main
39 } // end class Display

```

```

The character A has the value 65
The character B has the value 66
The character C has the value 67
The character a has the value 97
The character b has the value 98
The character c has the value 99
The character 0 has the value 48
The character 1 has the value 49
The character 2 has the value 50
The character $ has the value 36
The character * has the value 42
The character + has the value 43
The character / has the value 47
The character  has the value 32

```

3.28 (*Digits of an Integer*) Write an application that inputs one number consisting of five digits from the user, separates the number into its individual digits and displays the digits separated from one another by three spaces each. For example, if the user types in the number 42339, the application should display

```
4 2 3 3 9
```

Assume that the user enters the correct number of digits. What happens when you execute the application and type a number with more than five digits? What happens when you execute the application and type a number with fewer than five digits? [*Hint*: It is possible to do this exercise with the techniques you learned in this chapter. You'll need to use both division and remainder operations to "pick off" each digit.]

ANS: The last two sample outputs show the results of entering integers with fewer than five digits and more than five digits, respectively.

```

1  // Exercise 3.28 Solution: Five.cs
2  // Application breaks apart a five-digit number
3  using System;
4
5  public class Five
6  {
7      public static void Main( string[] args )
8      {
9          int number; // number input by user
10         int digit1; // first digit
11         int digit2; // second digit
12         int digit3; // third digit
13         int digit4; // fourth digit
14         int digit5; // fifth digit
15
16         Console.Write( "Enter five-digit integer: " ); // prompt
17         number = Convert.ToInt32( Console.ReadLine() ); // read number
18
19         // determine the five digits
20         digit1 = number / 10000;
21         digit2 = number % 10000 / 1000;
22         digit3 = number % 1000 / 100;
23         digit4 = number % 100 / 10;
24         digit5 = number % 10;
25
26         // output results
27         Console.WriteLine( "Digits in {0} are {1} {2} {3} {4} {5}",
28             number, digit1, digit2, digit3, digit4, digit5 );
29     } // end Main
30 } // end class Five

```

```

Enter five-digit integer: 12345
Digits in 12345 are 1 2 3 4 5

```

```

Enter five-digit integer: 123
Digits in 123 are 0 0 1 2 3

```

```

Enter five-digit integer: 7654321
Digits in 7654321 are 765 4 3 2 1

```

3.29 (*Table of Squares and Cubes*) Using only the programming techniques you learned in this chapter, write an application that calculates the squares and cubes of the numbers from 0 to 10 and displays the resulting values in table format, as shown below. All calculations should be done in terms of a variable *x*. [Note: This application does not require any input from the user.]

number	square	cube
0	0	0
1	1	1
2	4	8
3	9	27
4	16	64
5	25	125
6	36	216
7	49	343
8	64	512
9	81	729
10	100	1000

ANS:

```

1  // Exercise 3.29 Solution: Numbers.cs
2  // Application displays a table of squares and cubes
3  // of numbers from 0 to 10.
4  using System;
5
6  public class Numbers
7  {
8      public static void Main( string[] args )
9      {
10         // display a header for the table
11         Console.WriteLine( "{0}\t{1}\t{2}", "number", "square", "cube" );
12
13         // display x, x squared and x cubed for each value
14         int x = 0;
15         Console.WriteLine( "{0}\t{1}\t{2}", x, ( x * x ), ( x * x * x ) );
16         x = 1;
17         Console.WriteLine( "{0}\t{1}\t{2}", x, ( x * x ), ( x * x * x ) );
18         x = 2;
19         Console.WriteLine( "{0}\t{1}\t{2}", x, ( x * x ), ( x * x * x ) );
20         x = 3;
21         Console.WriteLine( "{0}\t{1}\t{2}", x, ( x * x ), ( x * x * x ) );
22         x = 4;
23         Console.WriteLine( "{0}\t{1}\t{2}", x, ( x * x ), ( x * x * x ) );
24         x = 5;
25         Console.WriteLine( "{0}\t{1}\t{2}", x, ( x * x ), ( x * x * x ) );
26         x = 6;
27         Console.WriteLine( "{0}\t{1}\t{2}", x, ( x * x ), ( x * x * x ) );
28         x = 7;
29         Console.WriteLine( "{0}\t{1}\t{2}", x, ( x * x ), ( x * x * x ) );
30         x = 8;
31         Console.WriteLine( "{0}\t{1}\t{2}", x, ( x * x ), ( x * x * x ) );
32         x = 9;
33         Console.WriteLine( "{0}\t{1}\t{2}", x, ( x * x ), ( x * x * x ) );
34         x = 10;
35         Console.WriteLine( "{0}\t{1}\t{2}", x, ( x * x ), ( x * x * x ) );
36     } // end Main
37 } // end class Numbers

```

number	square	cube
0	0	0
1	1	1
2	4	8
3	9	27
4	16	64
5	25	125
6	36	216
7	49	343
8	64	512
9	81	729
10	100	1000

3.30 (*Counting Negative, Positive and Zero Values*) Write an application that inputs five numbers and determines and displays the number of negative numbers input, the number of positive numbers input and the number of zeros input.

ANS:

```

1 // Exercise 3.30 Solution: Tally.cs
2 // Application accepts five numbers as input and displays a tally of the
3 // number of negatives, positives and zeros.
4 using System;
5
6 public class Tally
7 {
8     public static void Main( string[] args )
9     {
10         int inputNumber; // stores user input
11         int zeroTally; // count of 0s
12         int positiveTally; // count of positive values
13         int negativeTally; // count of negative values
14
15         // initialize counters
16         zeroTally = 0;
17         positiveTally = 0;
18         negativeTally = 0;
19
20         // prompt for input and read first number
21         Console.Write( "Enter first integer: " );
22         inputNumber = Convert.ToInt32( Console.ReadLine() );
23
24         // is the number 0?
25         if ( inputNumber == 0 )
26             zeroTally = zeroTally + 1;
27
28         // is the number negative?
29         if ( inputNumber < 0 )
30             negativeTally = negativeTally + 1;
31
32         // is the number positive?
33         if ( inputNumber > 0 )
34             positiveTally = positiveTally + 1;

```

```
35
36 // prompt for input and read second number
37 Console.Write( "Enter second integer: " );
38 inputNumber = Convert.ToInt32( Console.ReadLine() );
39
40 // is the number 0?
41 if ( inputNumber == 0 )
42     zeroTally = zeroTally + 1;
43
44 // is the number negative?
45 if ( inputNumber < 0 )
46     negativeTally = negativeTally + 1;
47
48 // is the number positive?
49 if ( inputNumber > 0 )
50     positiveTally = positiveTally + 1;
51
52 // prompt for input and read third number
53 Console.Write( "Enter third integer: " );
54 inputNumber = Convert.ToInt32( Console.ReadLine() );
55
56 // is the number 0?
57 if ( inputNumber == 0 )
58     zeroTally = zeroTally + 1;
59
60 // is the number negative?
61 if ( inputNumber < 0 )
62     negativeTally = negativeTally + 1;
63
64 // is the number positive?
65 if ( inputNumber > 0 )
66     positiveTally = positiveTally + 1;
67
68 // prompt for input and read fourth number
69 Console.Write( "Enter fourth integer: " );
70 inputNumber = Convert.ToInt32( Console.ReadLine() );
71
72 // is the number 0?
73 if ( inputNumber == 0 )
74     zeroTally = zeroTally + 1;
75
76 // is the number negative?
77 if ( inputNumber < 0 )
78     negativeTally = negativeTally + 1;
79
80 // is the number positive?
81 if ( inputNumber > 0 )
82     positiveTally = positiveTally + 1;
83
84 // prompt for input and read fifth number
85 Console.Write( "Enter fifth integer: " );
86 inputNumber = Convert.ToInt32( Console.ReadLine() );
87
```

```

88      // is the number 0?
89      if ( inputNumber == 0 )
90          zeroTally = zeroTally + 1;
91
92      // is the number negative?
93      if ( inputNumber < 0 )
94          negativeTally = negativeTally + 1;
95
96      // is the number positive?
97      if ( inputNumber > 0 )
98          positiveTally = positiveTally + 1;
99
100     // create a string describing the results
101     Console.WriteLine( "\nThere are {0} zeros", zeroTally );
102     Console.WriteLine( "There are {0} positive numbers",
103         positiveTally );
104     Console.WriteLine( "There are {0} negative numbers",
105         negativeTally );
106 } // end Main
107 } // end class Tally

```

```

Enter first integer: 0
Enter second integer: -7
Enter third integer: 3
Enter fourth integer: 13
Enter fifth integer: 5

There are 1 zeros
There are 3 positive numbers
There are 1 negative numbers

```

Making a Difference Exercises

3.31 (*Body Mass Index Calculator*) We introduced the body mass index (BMI) calculator in Exercise 1.19. The formulas for calculating the BMI are

$$BMI = \frac{\text{weightInPounds} \times 703}{\text{heightInInches} \times \text{heightInInches}}$$

or

$$BMI = \frac{\text{weightInKilograms}}{\text{heightInMeters} \times \text{heightInMeters}}$$

Create a BMI calculator application that reads the user's weight in pounds and height in inches (or, if you prefer, the user's weight in kilograms and height in meters), then calculates and displays the user's body mass index. The application should also display the following information from the Department of Health and Human Services/National Institutes of Health so the user can evaluate his/her BMI:

```

BMI VALUES
Underweight: less than 18.5
Normal:      between 18.5 and 24.9
Overweight:  between 25 and 29.9
Obese:       30 or greater

```


ANS:

```

1 // Exercise 3.31 Solution: BMICalculator.cs
2 // Calculate and display the user's body mass index.
3 using System;
4
5 public class BMICalculator
6 {
7     public static void Main( string[] args )
8     {
9         int weight; // weight in pounds
10        int height; // height in inches
11        int bmi; // user's body mass index
12
13        // obtain weight in pounds and height in inches
14        Console.WriteLine(
15            "Welcome to the body mass index (BMI) calculator" );
16        Console.Write( "Enter your weight in pounds: " );
17        weight = Convert.ToInt32( Console.ReadLine() );
18        Console.Write( "Enter your height in inches: " );
19        height = Convert.ToInt32( Console.ReadLine() );
20
21        // calculate BMI
22        bmi = ( weight * 703 ) / ( height * height );
23
24        // display results
25        Console.WriteLine( "Your BMI is: {0}\n", bmi );
26        Console.WriteLine( "BMI VALUES" );
27        Console.WriteLine( "Underweight: less than 18.5" );
28        Console.WriteLine( "Normal:      between 18.5 and 24.9" );
29        Console.WriteLine( "Overweight: between 25 and 29.9" );
30        Console.WriteLine( "Obese:      30 or greater" );
31    } // end Main
32 } // end class BMICalculator

```

```

Welcome to the body mass index (BMI) calculator
Enter your weight in pounds: 165
Enter your height in inches: 69
Your BMI is: 24

```

```

BMI VALUES
Underweight: less than 18.5
Normal:      between 18.5 and 24.9
Overweight:  between 25 and 29.9
Obese:       30 or greater

```

3.32 (Car-Pool Savings Calculator) Research several car-pooling websites. Create an application that calculates your daily driving cost, so that you can estimate how much money could be saved by car pooling, which also has other advantages such as reducing carbon emissions and reducing traffic congestion. The application should input the following information and display the user's cost per day of driving to work:

- a) Total miles driven per day.

- b) Cost per gallon of gasoline (in cents).
- c) Average miles per gallon.
- d) Parking fees per day (in cents).
- e) Tolls per day (in cents).

ANS:

```

1  // Exercise 3.32 Solution: DailyDrivingCost.cs
2  // Determine daily driving cost based on total miles driven per day,
3  // cost per gallon of gas, average miles per gallon, parking fees per day
4  // and tolls per day.
5  using System;
6
7  public class DailyDrivingCost
8  {
9      public static void Main( string[] args )
10     {
11         int milesDriven; // miles driven per day
12         int costPerGallon; // cost per gallon of gas (in cents)
13         int averageMilesPerGallon; // average miles per gallon of gas
14         int parkingFees; // parking fees per day
15         int tolls; // tolls per day
16
17         Console.WriteLine( "Welcome to the Daily Driving Cost calculator" );
18         Console.Write( "Enter the number of miles you drive per day: " );
19         milesDriven = Convert.ToInt32( Console.ReadLine() );
20         Console.Write( "Enter the cost per gallon of gas (in cents): " );
21         costPerGallon = Convert.ToInt32( Console.ReadLine() );
22         Console.Write( "Enter average miles per gallon of gas: " );
23         averageMilesPerGallon = Convert.ToInt32( Console.ReadLine() );
24         Console.Write( "Enter the parking fees per day (in cents): " );
25         parkingFees = Convert.ToInt32( Console.ReadLine() );
26         Console.Write( "Enter the tolls per day (in cents): " );
27         tolls = Convert.ToInt32( Console.ReadLine() );
28
29         // calculate and display daily driving cost
30         Console.WriteLine( "\nYour daily driving cost is {0}",
31             costPerGallon / 100.0 * milesDriven / averageMilesPerGallon +
32             parkingFees / 100.0 + tolls / 100.0 );
33     } // end Main
34 } // end class DailyDrivingCost

```

```

Welcome to the Daily Driving Cost calculator
Enter the number of miles you drive per day: 100
Enter the cost per gallon of gas (in cents): 291
Enter average miles per gallon of gas: 21
Enter the parking fees per day (in cents): 250
Enter the tolls per day (in cents): 300

```

```

Your daily driving cost is 19.3571428571429

```


Introduction to Classes and Objects

4

Nothing can have value without being an object of utility.

—Karl Marx

Your public servants serve you right.

—Adlai E. Stevenson

*Knowing how to answer one who speaks,
To reply to one who sends a message.*

—Amenemope

*You'll see something new.
Two things. And I call them
Thing One and Thing Two.*

—Dr. Theodor Seuss Geisel

Objectives

In this chapter you'll learn:

- How to declare a class and use it to create an object.
- How to implement a class's behaviors as methods.
- How to implement a class's attributes as instance variables and properties.
- How to call an object's methods to make them perform their tasks.
- What instance variables of a class and local variables of a method are.
- How to use a constructor to initialize an object's data.

Self-Review Exercises

4.1 Fill in the blanks in each of the following:

a) A house is to a blueprint as a(n) _____ is to a class.

ANS: object.

b) Every class declaration contains keyword _____ followed immediately by the class's name.

ANS: class.

c) Operator _____ creates an object of the class specified to the right of the keyword.

ANS: new.

d) Each parameter must specify both a(n) _____ and a(n) _____.

ANS: type, name.

e) By default, classes that are not explicitly declared in a namespace are implicitly placed in the _____.

ANS: global namespace.

f) When each object of a class maintains its own copy of an attribute, the field that represents the attribute is also known as a(n) _____.

ANS: instance variable.

g) C# provides three simple types for storing real numbers—_____, _____, and _____.

ANS: float, double, decimal.

h) Variables of type double represent _____ floating-point numbers.

ANS: double-precision.

i) Convert method _____ returns a decimal value.

ANS: ToDecimal.

j) Keyword public is a(n) _____.

ANS: access modifier.

k) Return type _____ indicates that a method will not return any information when it completes its task.

ANS: void.

l) Console method _____ reads characters until a newline character is encountered, then returns those characters (not including the newline) as a string.

ANS: ReadLine.

m) A(n) _____ is not required if you always refer to a class with its fully qualified class name.

ANS: using directive.

n) Variables of type float represent _____ floating-point numbers.

ANS: single-precision.

o) The format specifier _____ is used to display values in a monetary format.

ANS: C.

p) Types are either _____ types or _____ types.

ANS: value, reference.

q) For a(n) _____, the compiler automatically generates a private instance variable and set and get accessors.

ANS: auto-implemented property.

4.2 State whether each of the following is *true* or *false*. If *false*, explain why.

a) By convention, method names begin with a lowercase first letter and all subsequent words in the name begin with a capital first letter.

ANS: False. By convention, method names begin with an uppercase first letter and all subsequent words in the name begin with an uppercase first letter.

b) A property's get accessor enables a client to modify the value of the instance variable associated with the property.

ANS: False. A property's get accessor enables a client to retrieve the value of the instance variable associated with the property. A property's set accessor enables a client to modify the value of the instance variable associated with the property.

c) A using directive is not required when one class in a namespace uses another in the same namespace.

ANS: True.

d) Empty parentheses following a method name in a method declaration indicate that the method does not require any parameters to perform its task.

ANS: True.

e) After defining a property, you can use it the same way you use a method, but with empty parentheses, because no arguments are passed to a property.

ANS: False. After defining a property, you can use it the same way you use a variable.

f) Variables or methods declared with access modifier `private` are accessible only to methods and properties of the class in which they are declared.

ANS: True.

g) Variables declared in the body of a particular method are known as instance variables and can be used in all methods of the class.

ANS: False. Such variables are called local variables and can be used only in the method in which they are declared.

h) A property declaration must contain both a get accessor and a set accessor.

ANS: False. A property declaration can contain a get accessor, a set accessor or both.

i) The body of any method or property is delimited by left and right braces.

ANS: True.

j) Local variables are initialized by default.

ANS: False. Instance variables are initialized by default.

k) Reference-type instance variables are initialized by default to the value `null`.

ANS: True.

l) Any class that contains `public static void Main(string[] args)` can be used to execute an application.

ANS: True.

m) The number of arguments in the method call must match the number of required parameters in the method declaration's parameter list.

ANS: True.

n) Real-number values that appear in source code are known as floating-point literals and are of type `float` by default.

ANS: False. Such literals are of type `double` by default.

4.3 What is the difference between a local variable and an instance variable?

ANS: A local variable is declared in the body of a method and can be used only in the method in which it's declared. An instance variable is declared in a class, but not in the body of any of the class's methods. Every object (instance) of a class has a separate copy of the class's instance variables. Also, instance variables are accessible to all methods of the class. (We'll see an exception to this in Chapter 10, *Classes and Objects: A Deeper Look*.)

4 Chapter 4 Introduction to Classes and Objects

4.4 Explain the purpose of a method parameter. What is the difference between a parameter and an argument?

ANS: A parameter represents additional information that a method requires to perform its task. Each parameter required by a method is specified in the method's declaration. An argument is the actual value that is passed to a method parameter when a method is called.

Exercises

4.5 What is the purpose of operator `new`? Explain what happens when this keyword is used in an application.

ANS: The purpose of operator `new` is to create an object of a class. When operator `new` is used in an application, first a new object of the class to the right of `new` is created, then the class's constructor is called to ensure that the object is initialized properly.

4.6 What is a default constructor? How are an object's instance variables initialized if a class has only a default constructor?

ANS: A default constructor is a constructor provided by the compiler when you do not specify any constructors in the class. When a class has only the default constructor, its instance variables are initialized to their default values. Variables that contain numbers are initialized to 0, variables of type `bool` are initialized to `false`, and reference-type variables are initialized to `null`.

4.7 Explain the purpose of an instance variable.

ANS: A class provides an instance variable (or several instance variables) when each object of the class must maintain information separately from all other objects of the class. For example, a class called `Account` that represents a bank account provides an instance variable to represent the balance of the account. Each `Account` object maintains its own balance and does not know the balances of the bank's other accounts.

4.8 Explain how an application could use class `Console` without using a `using` directive.

ANS: If every use of a class's name in an application is fully qualified, there is no need for a `using` directive. A class's fully qualified name consists of the class's namespace followed by a dot and the class name. For example, an application could use class `Console` if every use of `Console` in the application were specified as `System.Console`.

4.9 Explain why a class might provide a property for an instance variable.

ANS: An instance variable is typically declared `private` in a class so that only the methods (and properties) of the class in which the instance variable is declared can manipulate the variable. In some cases, it may be necessary for an application to modify the `private` data. A class's designer can provide a `public` property that enables an application to specify the value for, or retrieve the value of, a `private` instance variable. The property's set accessor can ensure that the instance variable is set only to valid values. Using properties to access private fields allows the modification of the internal representation of the object without affecting the clients of the class.

4.10 Modify class `GradeBook` (Fig. 4.14) as follows:

- Include a second `string` auto-implemented property that represents the name of the course's instructor.
- Modify the constructor to specify two parameters—one for the course name and one for the instructor's name.
- Modify method `DisplayMessage` such that it first outputs the welcome message and course name, then outputs "This course is presented by: ", followed by the instructor's name.

Use your modified class in a test application that demonstrates the class's new capabilities.

ANS:

```

1 // Exercise 4.10 Solution: GradeBook.cs
2 // GradeBook class with a constructor to initialize the course name
3 // and instructor name.
4 using System;
5
6 public class GradeBook
7 {
8     // auto-implemented property CourseName implicitly creates an
9     // instance variable for this GradeBook's course name
10    public string CourseName { get; set; }
11
12    // auto-implemented property InstructorName implicitly creates an
13    // instance variable for this course's instructor
14    public string InstructorName { get; set; }
15
16    // constructor initializes automatic properties
17    // with two strings supplied as arguments
18    public GradeBook( string course, string instructor )
19    {
20        CourseName = course; // set CourseName to course
21        InstructorName = instructor; // set InstructorName to instructor
22    } // end constructor
23
24    // display a welcome message to the GradeBook user
25    public void DisplayMessage()
26    {
27        // this statement uses automatic properties CourseName and
28        // InstructorName to get the name of the course and instructor
29        Console.WriteLine( "Welcome to the grade book for\n{0}!",
30            CourseName );
31        Console.WriteLine( "This course is presented by: {0}",
32            InstructorName );
33    } // end method DisplayMessage
34 } // end class GradeBook

```

```

1 // Exercise 4.10 Solution: GradeBookTest.cs
2 // GradeBook constructor used to specify the course name
3 // and instructor name at the time each GradeBook object is created.
4 using System;
5
6 public class GradeBookTest
7 {
8     // Main method begins program execution
9     public static void Main( string[] args )
10    {
11        // create GradeBook object
12        GradeBook gradeBook1 = new GradeBook(
13            "CS101 Introduction to C# Programming", "Sam Smith" );
14    }

```

```

15     gradeBook1.DisplayMessage(); // display welcome message
16
17     Console.WriteLine( "\nChanging instructor name to Judy Jones\n" );
18     gradeBook1.InstructorName = "Judy Jones";
19
20     gradeBook1.DisplayMessage(); // display welcome message
21 } // end Main
22 } // end class GradeBookTest

```

```

Welcome to the grade book for
CS101 Introduction to C# Programming!
This course is presented by: Sam Smith

Changing instructor name to Judy Jones

Welcome to the grade book for
CS101 Introduction to C# Programming!
This course is presented by: Judy Jones

```

4.11 (Account Modification) Modify class `Account` (Fig. 4.15) to provide a method called `Debit` that withdraws money from an `Account`. Ensure that the debit amount doesn't exceed the balance. If it does, the balance should not be changed and the method should display a message indicating "Debit amount exceeded account balance." Modify class `AccountTest` (Fig. 4.16) to test method `Debit`.

ANS:

```

1  // Exercise 4.11 Solution: Account.cs
2  // Account class with a Debit method
3  // that withdraws money from the account.
4  using System;
5
6  public class Account
7  {
8      private decimal balance; // instance variable that stores the balance
9
10     // constructor
11     public Account( decimal initialBalance )
12     {
13         Balance = initialBalance; // set balance using property
14     } // end Account constructor
15
16     // credits (adds) an amount to the account
17     public void Credit( decimal amount )
18     {
19         Balance = Balance + amount; // add amount to balance
20     } // end method Credit
21
22     // debits (subtracts) an amount from the account
23     public void Debit( decimal amount )
24     {
25         if ( amount > Balance )
26             Console.WriteLine( "Debit amount exceeded account balance." );
27

```

```

28     Balance = Balance - amount; // subtract amount from balance
29 } // end method Debit
30
31 // property to get the balance
32 public decimal Balance
33 {
34     get
35     {
36         return balance;
37     } // end get
38     set
39     {
40         // validate that value is greater than or equal to 0;
41         // if it is not, balance is left unchanged
42         if ( value >= 0 )
43             balance = value;
44     } // end set
45 } // end property Balance
46 } // end class Account

```

```

1 // Exercise 4.11 Solution: AccountTest.cs
2 // Create and manipulate Account objects with a Debit method.
3 using System;
4
5 public class AccountTest
6 {
7     // Main method begins execution of C# application
8     public static void Main( string[] args )
9     {
10         Account account1 = new Account( 50.00M ); // create Account object
11         Account account2 = new Account( -7.53M ); // create Account object
12
13         // display initial balance of each object
14         Console.WriteLine( "account1 balance: {0:C}",
15             account1.Balance );
16         Console.WriteLine( "account2 balance: {0:C}\n",
17             account2.Balance );
18
19         decimal withdrawalAmount; // withdrawal amount entered by user
20
21         Console.Write( "Enter withdrawal amount for account1: " );
22         // obtain user input
23         withdrawalAmount = Convert.ToDecimal( Console.ReadLine() );
24         Console.WriteLine( "\nsubtracting {0:C} from account1 balance",
25             withdrawalAmount );
26         // subtract amount from account1
27         account1.Debit( withdrawalAmount );
28
29         // display balances
30         Console.WriteLine( "account1 balance: {0:C}",
31             account1.Balance );
32         Console.WriteLine( "account2 balance: {0:C}\n",
33             account2.Balance );

```

```

34
35     Console.Write( "Enter withdrawal amount for account2: " );
36     // obtain user input
37     withdrawalAmount = Convert.ToDecimal( Console.ReadLine() );
38     Console.WriteLine( "\nsubtracting {0:C} from account2 balance",
39         withdrawalAmount );
40     // subtract amount from account2
41     account2.Debit( withdrawalAmount );
42
43     // display balances
44     Console.WriteLine( "account1 balance: {0:C}",
45         account1.Balance );
46     Console.WriteLine( "account2 balance: {0:C}",
47         account2.Balance );
48 } // end Main
49 } // end class AccountTest

```

```

account1 balance: $50.00
account2 balance: $0.00

Enter withdrawal amount for account1: 24.99

subtracting $24.99 from account1 balance
account1 balance: $25.01
account2 balance: $0.00

Enter withdrawal amount for account2: 24.99

subtracting $24.99 from account2 balance
Debit amount exceeded account balance.
account1 balance: $25.01
account2 balance: $0.00

```

4.12 (Invoice Class) Create a class called *Invoice* that a hardware store might use to represent an invoice for an item sold at the store. An *Invoice* should include four pieces of information as either instance variables or automatic properties—a part number (type *string*), a part description (type *string*), a quantity of the item being purchased (type *int*) and a price per item (decimal). Your class should have a constructor that initializes the four values. Provide a property with a *get* and *set* accessor for any instance variables. For the *Quantity* and *PricePerItem* properties, if the value passed to the *set* accessor is negative, the value of the instance variable should be left unchanged. Also, provide a method named *GetInvoiceAmount* that calculates the invoice amount (i.e., multiplies the quantity by the price per item), then returns the amount as a decimal value. Write a test application named *InvoiceTest* that demonstrates class *Invoice*'s capabilities.

ANS:

```

1 // Exercise 4.12 Solution: Invoice.cs
2 // Invoice class.
3
4 public class Invoice
5 {
6     // auto-implemented property implicitly creates an
7     // instance variable for the part number
8     public string PartNumber { get; set; }

```

```
9
10 // auto-implemented property implicitly creates an
11 // instance variable for the part description
12 public string PartDescription { get; set; }
13
14 private int quantity;
15 private decimal pricePerItem;
16
17 // four-parameter constructor
18 public Invoice( string part, string description, int count,
19             decimal price )
20 {
21     // initialize automatic properties and instance variables
22     PartNumber = part;
23     PartDescription = description;
24     Quantity = count;
25     PricePerItem = price;
26 } // end four-parameter Invoice constructor
27
28 // property to get and set the quantity
29 public int Quantity
30 {
31     get
32     {
33         return quantity;
34     } // end get
35     set
36     {
37         if ( value >= 0 ) // determine whether count is positive
38             quantity = value; // valid count assigned to quantity
39     } // end set
40 } // end property Quantity
41
42 // property to get and set the price per item
43 public decimal PricePerItem
44 {
45     get
46     {
47         return pricePerItem;
48     } // end get
49     set
50     {
51         if ( value >= 0 ) // determine whether price is nonnegative
52             pricePerItem = value; // valid price assigned to pricePerItem
53     } // end set
54 } // end property PricePerItem
55
56 // calculates and returns the invoice amount
57 public decimal GetInvoiceAmount()
58 {
59     return Quantity * PricePerItem; // calculate total cost
60 } // end method GetInvoiceAmount
61 } // end class Invoice
```

```
1 // Exercise 4.12 Solution: InvoiceTest.cs
2 // Application to test class Invoice.
3 using System;
4
5 public class InvoiceTest
6 {
7     public static void Main( string[] args )
8     {
9         Invoice invoice1 = new Invoice( "1234", "Hammer", 2, 14.95M );
10
11         // display invoice1
12         Console.WriteLine( "Original invoice information" );
13         Console.WriteLine( "Part number: {0}", invoice1.PartNumber );
14         Console.WriteLine( "Description: {0}",
15             invoice1.PartDescription );
16         Console.WriteLine( "Quantity: {0}", invoice1.Quantity );
17         Console.WriteLine( "Price: {0:C}", invoice1.PricePerItem );
18         Console.WriteLine( "Invoice amount: {0:C}",
19             invoice1.GetInvoiceAmount() );
20
21         // change invoice1's data
22         invoice1.PartNumber = "001234";
23         invoice1.PartDescription = "Yellow Hammer";
24         invoice1.Quantity = 3;
25         invoice1.PricePerItem = 19.49M;
26
27         // display invoice1 with new data
28         Console.WriteLine( "\nUpdated invoice information" );
29         Console.WriteLine( "Part number: {0}", invoice1.PartNumber );
30         Console.WriteLine( "Description: {0}",
31             invoice1.PartDescription );
32         Console.WriteLine( "Quantity: {0}", invoice1.Quantity );
33         Console.WriteLine( "Price: {0:C}", invoice1.PricePerItem );
34         Console.WriteLine( "Invoice amount: {0:C}",
35             invoice1.GetInvoiceAmount() );
36
37         Invoice invoice2 = new Invoice( "5678", "PaintBrush", -5, -9.99M );
38
39         // display invoice2
40         Console.WriteLine( "\nOriginal invoice information" );
41         Console.WriteLine( "Part number: {0}", invoice2.PartNumber );
42         Console.WriteLine( "Description: {0}",
43             invoice2.PartDescription );
44         Console.WriteLine( "Quantity: {0}", invoice2.Quantity );
45         Console.WriteLine( "Price: {0:C}", invoice2.PricePerItem );
46         Console.WriteLine( "Invoice amount: {0:C}",
47             invoice2.GetInvoiceAmount() );
48
49         // change invoice2's data
50         invoice2.Quantity = 3;
51         invoice2.PricePerItem = 9.49M;
52
53         // display invoice2 with new data
54         Console.WriteLine( "\nUpdated invoice information" );
```

```

55     Console.WriteLine( "Part number: {0}", invoice2.PartNumber );
56     Console.WriteLine( "Description: {0}",
57         invoice2.PartDescription );
58     Console.WriteLine( "Quantity: {0}", invoice2.Quantity );
59     Console.WriteLine( "Price: {0:C}", invoice2.PricePerItem );
60     Console.WriteLine( "Invoice amount: {0:C}",
61         invoice2.GetInvoiceAmount() );
62 } // end Main
63 } // end class InvoiceTest

```

```

Original invoice information
Part number: 1234
Description: Hammer
Quantity: 2
Price: $14.95
Invoice amount: $29.90

```

```

Updated invoice information
Part number: 001234
Description: Yellow Hammer
Quantity: 3
Price: $19.49
Invoice amount: $58.47

```

```

Original invoice information
Part number: 5678
Description: PaintBrush
Quantity: 0
Price: $0.00
Invoice amount: $0.00

```

```

Updated invoice information
Part number: 5678
Description: PaintBrush
Quantity: 3
Price: $9.49
Invoice amount: $28.47

```

4.13 (*Employee Class*) Create a class called `Employee` that includes three pieces of information as either instance variables or automatic properties—a first name (type `string`), a last name (type `string`) and a monthly salary (decimal). Your class should have a constructor that initializes the three values. Provide a property with a get and set accessor for any instance variables. If the monthly salary is negative, the set accessor should leave the instance variable unchanged. Write a test application named `EmployeeTest` that demonstrates class `Employee`'s capabilities. Create two `Employee` objects and display each object's *yearly* salary. Then give each `Employee` a 10% raise and display each `Employee`'s yearly salary again.

ANS:

```

1 // Exercise 4.13 Solution: Employee.cs
2 // Employee class.
3
4 public class Employee
5 {

```

```

6    // auto-implemented property FirstName implicitly creates an
7    // instance variable for the employee's first name
8    public string FirstName { get; set; }
9
10   // auto-implemented property LastName implicitly creates an
11   // instance variable for the employee's last name
12   public string LastName { get; set; }
13
14   private decimal monthlySalary;
15
16   // constructor to initialize first name, last name and monthly salary
17   public Employee( string first, string last, decimal salary )
18   {
19       FirstName = first;
20       LastName = last;
21       MonthlySalary = salary;
22   } // end three-parameter Employee constructor
23
24   // property to get and set the salary
25   public decimal MonthlySalary
26   {
27       get
28       {
29           return monthlySalary;
30       } // end get
31       set
32       {
33           if ( value >= 0 ) // determine whether salary is nonnegative
34               monthlySalary = value;
35       } // end set
36   } // end property MonthlySalary
37 } // end class Employee

```

```

1  // Exercise 4.13 Solution: EmployeeTest.cs
2  // Application to test class Employee.
3  using System;
4
5  public class EmployeeTest
6  {
7      public static void Main( string[] args )
8      {
9          Employee employee1 = new Employee( "Bob", "Jones", 2875.00M );
10         Employee employee2 = new Employee( "Susan", "Baker", 3150.75M );
11
12         // display employees
13         Console.WriteLine( "Employee 1: {0} {1}; Yearly Salary: {2:C}",
14             employee1.FirstName, employee1.LastName,
15             12 * employee1.MonthlySalary );
16         Console.WriteLine( "Employee 2: {0} {1}; Yearly Salary: {2:C}",
17             employee2.FirstName, employee2.LastName,
18             12 * employee2.MonthlySalary );
19     }

```

```

20      // increase employee salaries by 10%
21      Console.WriteLine( "\nIncreasing employee salaries by 10%" );
22      employee1.MonthlySalary = employee1.MonthlySalary * 1.10M;
23      employee2.MonthlySalary = employee2.MonthlySalary * 1.10M;
24
25      // display employees with new yearly salary
26      Console.WriteLine( "Employee 1: {0} {1}; Yearly Salary: {2:C}",
27          employee1.FirstName, employee1.LastName,
28          12 * employee1.MonthlySalary );
29      Console.WriteLine( "Employee 2: {0} {1}; Yearly Salary: {2:C}",
30          employee2.FirstName, employee2.LastName,
31          12 * employee2.MonthlySalary );
32  } // end Main
33 } // end class EmployeeTest

```

```

Employee 1: Bob Jones; Yearly Salary: $34,500.00
Employee 2: Susan Baker; Yearly Salary: $37,809.00

```

```

Increasing employee salaries by 10%
Employee 1: Bob Jones; Yearly Salary: $37,950.00
Employee 2: Susan Baker; Yearly Salary: $41,589.90

```

4.14 (Date Class) Create a class called *Date* that includes three pieces of information as automatic properties—a month (type *int*), a day (type *int*) and a year (type *int*). Your class should have a constructor that initializes the three automatic properties and assumes that the values provided are correct. Provide a method *DisplayDate* that displays the month, day and year separated by forward slashes (/). Write a test application named *DateTest* that demonstrates class *Date*'s capabilities.

ANS:

```

1  // Exercise 4.14 Solution: Date.cs
2  // Date class with automatic properties for the month, day and year.
3  using System;
4
5  public class Date
6  {
7      // auto-implemented property Month implicitly creates an
8      // instance variable for the month
9      public int Month { get; set; }
10
11     // auto-implemented property Day implicitly creates an
12     // instance variable for the day
13     public int Day { get; set; }
14
15     // auto-implemented property Year implicitly creates an
16     // instance variable for the year
17     public int Year { get; set; }
18
19     // constructor
20     public Date( int monthValue, int dayValue, int yearValue )
21     {
22         Month = monthValue;

```

```

23     Day = dayValue;
24     Year = yearValue;
25 } // end three-parameter constructor
26
27 // display the date
28 public void DisplayDate()
29 {
30     Console.Write( "{0}/{1}/{2}", Month, Day, Year );
31 } // end method DisplayDate
32 } // end class Date

```

```

1 // Exercise 4.14 Solution: DateTest.cs
2 // Application to test class Date.
3 using System;
4
5 public class DateTest
6 {
7     public static void Main( string[] args )
8     {
9         Date date1 = new Date( 7, 4, 2004 );
10
11         Console.Write( "The initial date is: " );
12         date1.DisplayDate();
13
14         // change date values
15         date1.Month = 11;
16         date1.Day = 1;
17         date1.Year = 2003;
18
19         Console.Write( "\nDate with new values is: " );
20         date1.DisplayDate();
21
22         Console.WriteLine(); // output a new line
23     } // end Main
24 } // end class DateTest

```

```

The initial date is: 7/4/2004
Date with new values is: 11/1/2003

```

Making a Difference Exercises

4.15 (*Target-Heart-Rate Calculator*) While exercising, you can use a heart-rate monitor to see that your heart rate stays within a safe range suggested by your trainers and doctors. According to the American Heart Association (AHA) (www.americanheart.org/presenter.jhtml?identifier=4736), the formula for calculating your *maximum heart rate* in beats per minute is 220 minus your age in years. Your *target heart rate* is a range that is 50–85% of your maximum heart rate. [Note: These formulas are estimates provided by the AHA. Maximum and target heart rates may vary based on the health, fitness and gender of the individual. Always consult a physician or qualified health care professional before beginning or modifying an exercise program.] Create a class called `HeartRates`. The class attributes should include the person's first name, last name, year of birth and the current year. Your class should have a constructor that receives this data as parameters. For each attribute provide

a property with set and get accessors. The class also should include a property that calculates and returns the person's age (in years), a property that calculates and returns the person's maximum heart rate and properties that calculate and return the person's minimum and maximum target heart rates. Write an application that prompts for the person's information, instantiates an object of class HeartRates and prints the information from that object—including the person's first name, last name and year of birth—then calculates and prints the person's age in (years), maximum heart rate and target-heart-rate range.

ANS:

```

1 // Exercise 4.15 Solution: HeartRates.cs
2 // Maintains information about a person's maximum and target heart rates.
3 // This class assumes that the birth and current years are correct values.
4 public class HeartRates
5 {
6     public string FirstName { get; set; }
7     public string LastName { get; set; }
8     public int BirthYear { get; set; }
9     public int CurrentYear { get; set; }
10
11     // constructor
12     public HeartRates( string firstName, string lastName,
13         int birthYear, int currentYear )
14     {
15         FirstName = firstName;
16         LastName = lastName;
17         BirthYear = birthYear;
18         CurrentYear = currentYear;
19     } // end constructor
20
21     // returns the person's age in years
22     public int Age
23     {
24         get
25         {
26             return CurrentYear - BirthYear;
27         } // end get
28     } // end property Age
29
30     // returns the maximum heart rate
31     public int MaximumHeartRate
32     {
33         get
34         {
35             return 220 - Age;
36         } // end get
37     } // end property MaximumHeartRate
38
39     // returns the minimum target heart rate (50% of maximum heart rate)
40     public double MinimumTargetHeartRate
41     {
42         get
43         {
44             return .5 * MaximumHeartRate;
45         } // end get

```

```

46     } // end property MinimumTargetHeartRate
47
48     // returns the maximum target heart rate (85% of maximum heart rate)
49     public double MaximumTargetHeartRate
50     {
51         get
52         {
53             return .85 * MaximumHeartRate;
54         } // end get
55     } // end property MaximumTargetHeartRate
56 } // end class HeartRates

```

```

1  // Exercise 4.15 Solution: HeartRatesTest.cs
2  // Tests class HeartRates.
3  using System;
4
5  public class HeartRatesTest
6  {
7      public static void Main( string[] args )
8      {
9          // create a HeartRates object for a person born in 1975;
10         // rates calculated based on current year 2010.
11         HeartRates bob = new HeartRates( "Bob", "Blue", 1975, 2010 );
12
13         Console.WriteLine( "First name: {0}", bob.FirstName );
14         Console.WriteLine( "Last name: {0}", bob.LastName );
15         Console.WriteLine( "Age: {0}", bob.Age );
16         Console.WriteLine( "Maximum heart rate: {0}", bob.MaximumHeartRate );
17         Console.WriteLine( "Target heart rate range:" );
18         Console.WriteLine( "    Minimum: {0}", bob.MinimumTargetHeartRate );
19         Console.WriteLine( "    Maximum: {0}", bob.MaximumTargetHeartRate );
20     } // end Main
21 } // end class HeartRatesTest

```

```

First name: Bob
Last name: Blue
Age: 35
Maximum heart rate: 185
Target heart rate range:
    Minimum: 92.5
    Maximum: 157.25

```

4.16 (*Computerization of Health Records*) A health care issue that has been in the news lately is the computerization of health records. This possibility is being approached cautiously because of sensitive privacy and security concerns, among others. [We address such concerns in later exercises.] Computerizing health records could make it easier for patients to share their health profiles and histories among their various health care professionals. This could improve the quality of health care, help avoid drug conflicts and erroneous drug prescriptions, reduce costs and, in emergencies, could save lives. In this exercise, you'll design a "starter" `HealthProfile` class for a person. The class attributes should include the person's first name, last name, gender, year of birth, height (in inches) and weight (in pounds). Your class should have a constructor that receives this data. For each attribute

provide a property with set and get accessors. The class also should include methods that calculate and return the user's age in years, maximum heart rate and target-heart-rate range (see Exercise 4.15), and body mass index (BMI; see Exercise 3.31). Write an application that prompts for the person's information, instantiates an object of class `HealthProfile` for that person and prints the information from that object—including the person's first name, last name, gender, date of birth, height and weight—then calculates and prints the person's age in years, BMI, maximum heart rate and target-heart-rate range. It should also display the “BMI values” chart from Exercise 3.31.

ANS:

```

1 // Exercise 4.16 Solution: HealthProfile.cs
2 // Maintains information about a person's maximum and target hear rates.
3 // This class assumes values passed to the set accessors are correct.
4 using System;
5
6 public class HealthProfile
7 {
8     public string FirstName { set; get; }
9     public string LastName { set; get; }
10    public string Gender { set; get; }
11    public int BirthYear { set; get; }
12    public int CurrentYear { set; get; }
13    public double Height { set; get; }
14    public double Weight { set; get; }
15
16    // constructor
17    public HealthProfile( string firstName, string lastName, string gender,
18        double height, double weight, int birthYear, int currentYear)
19    {
20        FirstName = firstName;
21        LastName = lastName;
22        Gender = gender;
23        Height = height;
24        Weight = weight;
25        BirthYear = birthYear;
26        CurrentYear = currentYear;
27    } // end constructor
28
29    // returns the person's age in years
30    public int Age
31    {
32        get
33        {
34            return CurrentYear - BirthYear;
35        } // end get
36    } // end property Age
37
38    // returns the maximum heart rate
39    public int MaximumHeartRate
40    {
41        get
42        {
43            return 220 - Age;
44        } // end get
45    } // end property MaximumHeartRate

```

```

46
47 // returns the minimum target heart rate (50% of maximum heart rate)
48 public double MinimumTargetHeartRate
49 {
50     get
51     {
52         return .5 * MaximumHeartRate;
53     } // end get
54 } // end property MinimumTargetHeartRate
55
56 // returns the maximum target heart rate (85% of maximum heart rate)
57 public double MaximumTargetHeartRate
58 {
59     get
60     {
61         return .85 * MaximumHeartRate;
62     } // end get
63 } // end property MaximumTargetHeartRate
64
65 // returns the person's body mass index
66 public double BMI
67 {
68     get
69     {
70         return ( Weight * 703 ) / ( Height * Height );
71     } // end get
72 } // end property BMI
73
74 // displays the person's health profile
75 public void DisplayHealthProfile()
76 {
77     Console.WriteLine( "\nHEALTH PROFILE FOR: {0} {1}\n",
78         FirstName, LastName );
79     Console.WriteLine( "Gender: {0}", Gender );
80     Console.WriteLine( "Age: {0}", Age );
81     Console.WriteLine( "Height (in inches): {0}", Height );
82     Console.WriteLine( "Weight (in pounds): {0}", Weight );
83     Console.WriteLine( "Maximum heart rate: {0}", MaximumHeartRate );
84     Console.WriteLine( "Target heart rate range:" );
85     Console.WriteLine( "    Minimum: {0}", MinimumTargetHeartRate );
86     Console.WriteLine( "    Maximum: {0}", MaximumTargetHeartRate );
87     Console.WriteLine( "BMI: {0}\n", BMI );
88     Console.WriteLine( "BMI VALUES" );
89     Console.WriteLine( "Underweight: less than 18.5" );
90     Console.WriteLine( "Normal:      between 18.5 and 24.9" );
91     Console.WriteLine( "Overweight: between 25 and 29.9" );
92     Console.WriteLine( "Obese:      30 or greater" );
93 } // end method DisplayHealthProfile
94 } // end class HealthProfile

```

```

1 // Exercise 4.16 Solution: HealthProfileTest.cs
2 // Tests class HealthProfile.
3 using System;

```

```

4
5 public class HealthProfileTest
6 {
7     public static void Main( string[] args )
8     {
9         Console.Write( "Enter first name: " );
10        string firstName = Console.ReadLine();
11        Console.Write( "Enter last name: " );
12        string lastName = Console.ReadLine();
13        Console.Write( "Enter gender: " );
14        string gender = Console.ReadLine();
15        Console.Write( "Enter height in inches: " );
16        double height = Convert.ToDouble( Console.ReadLine() );
17        Console.Write( "Enter weight in pounds: " );
18        double weight = Convert.ToDouble( Console.ReadLine() );
19        Console.Write( "Enter year of birth: " );
20        int birthYear = Convert.ToInt32( Console.ReadLine() );
21        Console.Write( "Enter current year: " );
22        int currentYear = Convert.ToInt32( Console.ReadLine() );
23
24        // create a HealthProfile object for a person based on the user input
25        HealthProfile person =
26            new HealthProfile( firstName, lastName, gender, height,
27                weight, birthYear, currentYear );
28
29        // display user's health profile
30        person.DisplayHealthProfile();
31    } // end Main
32 } // end class HealthProfileTest

```

```

Enter first name: Jessica
Enter last name: Pink
Enter gender: F
Enter height in inches: 64
Enter weight in pounds: 120
Enter year of birth: 1975
Enter current year: 2010

HEALTH PROFILE FOR: Jessica Pink

Gender: F
Age: 35
Height (in inches): 64
Weight (in pounds): 120
Maximum heart rate: 185
Target heart rate range:
    Minimum: 92.5
    Maximum: 157.25
BMI: 20.595703125

BMI VALUES
Underweight: less than 18.5
Normal:      between 18.5 and 24.9
Overweight:  between 25 and 29.9
Obese:       30 or greater

```


Control Statements: Part I

Solutions

5

Let's all move one place on.

—Lewis Carroll

The wheel is come full circle.

—William Shakespeare

*How many apples fell on
Newton's head before he took the
hint!*

—Robert Frost

*All the evolution we know of
proceeds from the vague to the
definite.*

—Charles Sanders Peirce

Objectives

In this chapter you'll learn:

- Basic problem solving techniques.
- To develop algorithms through the process of top-down, stepwise refinement.
- To use the `if` and `if...else` selection statements to choose between actions.
- To use the `while` statement to execute statements in an application repeatedly.
- To use counter-controlled repetition and sentinel-controlled repetition.
- To use the increment, decrement and compound assignment operators.

Self-Review Exercises

5.1 Fill in the blanks in each of the following statements:

- a) All applications can be written in terms of three types of control structures: _____, _____ and _____.

ANS: sequence, selection, repetition.

- b) The _____ statement is used to execute one action when a condition is true and another when that condition is false.

ANS: `if...else`.

- c) Repeating a set of instructions a specific number of times is called _____ repetition.

ANS: counter-controlled (or definite).

- d) When it is not known in advance how many times a set of statements will be repeated, a(n) _____ value can be used to terminate the repetition.

ANS: sentinel, signal, flag or dummy.

- e) The _____ structure is built into C#—by default, statements execute in the order they appear.

ANS: sequence.

- f) Instance variables of type `int` are given the value _____ by default.

ANS: 0 (zero).

- g) C# is a _____ language—it requires all variables to have a type.

ANS: strongly typed.

- h) If the increment operator is _____ to a variable, the variable is incremented by 1 first, then its new value is used in the expression.

ANS: prefixed.

5.2 State whether each of the following is *true* or *false*. If *false*, explain why.

- a) An algorithm is a procedure for solving a problem in terms of the actions to execute and the order in which these actions execute.

ANS: True.

- b) A set of statements contained within a pair of parentheses is called a block.

ANS: False. A set of statements contained within a pair of braces (`{` and `}`) is called a block.

- c) A selection statement specifies that an action is to be repeated while some condition remains true.

ANS: False. A repetition statement specifies that an action is to be repeated while some condition remains true. A selection statement determines whether an action is performed based on the truth or falsity of a condition.

- d) A nested control statement appears in the body of another control statement.

ANS: True.

- e) C# provides the arithmetic compound assignment operators `+=`, `-=`, `*=`, `/=` and `%=` for abbreviating assignment expressions.

ANS: True.

- f) Specifying the order in which statements (actions) execute in an application is called program control.

ANS: True.

- g) The unary cast operator (`double`) creates a temporary integer copy of its operand.

ANS: False. The unary cast operator (`double`) creates a temporary floating-point copy of its operand.

- h) Instance variables of type `bool` are given the value `true` by default.

ANS: False. Instance variables of type `bool` are given the value `false` by default.

- i) Pseudocode helps you think out an application before attempting to write it in a programming language.

ANS: True.

- 5.3** Write four different C# statements that each add 1 to integer variable x.

ANS: `x = x + 1;`
`x += 1;`
`++x;`
`x++;`

- 5.4** Write C# statements to accomplish each of the following tasks:

- a) Assign the sum of x and y to z, and increment x by 1 after the calculation. Use only one statement.

ANS: `z = x++ + y;`

- b) Test whether variable count is greater than 10. If it is, display "Count is greater than 10".

ANS: `if (count > 10)`
`Console.WriteLine("Count is greater than 10");`

- c) Decrement the variable x by 1, then subtract it from the variable total. Use only one statement.

ANS: `total -= --x;`

- d) Calculate the remainder after q is divided by divisor, and assign the result to q. Write this statement in two different ways.

ANS: `q %= divisor;`
`q = q % divisor;`

- 5.5** Write a C# statement to accomplish each of the following tasks:

- a) Declare variable sum to be of type int.

ANS: `int sum;`

- b) Declare variable x to be of type int.

ANS: `int x;`

- c) Assign 1 to variable x.

ANS: `x = 1;`

- d) Assign 0 to variable sum.

ANS: `sum = 0;`

- e) Add variable x to variable sum, and assign the result to variable sum.

ANS: `sum += x;` or `sum = sum + x;`

- f) Display "The sum is: ", followed by the value of variable sum.

ANS: `Console.WriteLine("The sum is: {0}", sum);`

- 5.6** Combine the statements that you wrote in Exercise 5.5 into a C# application that calculates and displays the sum of the integers from 1 to 10. Use a while statement to loop through the calculation and increment statements. The loop should terminate when the value of x becomes 11.

ANS: The application is as follows:

```

1 // Exercise 5.6 Solution: Calculate.cs
2 // Calculate the sum of the integers from 1 to 10
3 using System;
4
5 public class Calculate
6 {
7     public static void Main( string[] args )
8     {
9         int sum;
```

```

10     int x;
11
12     x = 1; // initialize x to 1 for counting
13     sum = 0; // initialize sum to 0 for totaling
14
15     while ( x <= 10 ) // while x is less than or equal to 10
16     {
17         sum += x; // add x to sum
18         ++x; // increment x
19     } // end while
20
21     Console.WriteLine( "The sum is: {0}", sum );
22 } // end Main
23 } // end class Calculate

```

The sum is: 55

5.7 Determine the values of the variables in the following statement after it executes. Assume that when the statement begins executing, all variables are type `int` and have the value 5.

```
product *= x++;
```

ANS: product = 25, x = 6

5.8 Identify and correct the errors in each of the following sets of code:

a) `while (c <= 5)`

```

{
    product *= c;
    ++c;

```

ANS: Error: The closing right brace of the `while` statement's body is missing.

Correction: Add a closing right brace after the statement `++c;`.

b) `if (gender == 1)`

```

    Console.WriteLine( "Woman" );
else;

```

```
    Console.WriteLine( "Man" );
```

ANS: Error: The semicolon after `else` results in a logic error. The second output statement will always execute.

Correction: Remove the semicolon after `else`.

5.9 What is wrong with the following `while` statement?

```

while ( z >= 0 )
    sum += z;

```

ANS: The value of the variable `z` is never changed in the `while` statement. Therefore, an infinite loop occurs if the loop-continuation condition (`z >= 0`) is initially true. To prevent an infinite loop, `z` must be decremented so that it eventually becomes less than 0.

Exercises

5.10 Compare and contrast the `if` single-selection statement and the `while` repetition statement. How are these two statements similar? How are they different?

ANS: The `if` single-selection statement and the `while` repetition statement both perform an action (or set of actions) based on whether a condition is true or false. However, if the condition is true, the `if` single-selection statement performs the action(s) once, whereas the `while` repetition statement repeatedly performs the action(s) until the condition becomes false.

5.11 Explain what happens when a C# application attempts to divide one integer by another integer. What happens to the fractional part of the calculation? How can you avoid such an outcome?

ANS: Dividing two integers results in integer division—any fractional part of the calculation is lost (i.e., truncated). For example, $7 \div 4$, which yields 1.75 in conventional arithmetic, truncates to 1 in integer arithmetic, rather than rounding to 2. To obtain a floating-point result from dividing integer values, you must temporarily treat these values as floating-point numbers in the calculation by using the unary cast operator (`double`). As long as the (`double`) cast operator is applied to any variable in the calculation, the calculation will yield a `double` result which can be assigned to a `double` variable.

5.12 Describe the two ways in which control statements can be combined.

ANS: Control statements can be “attached” (that is, stacked) to one another by connecting the exit point of one to the entry point of the next. Control statements also can be nested by placing one control statement inside another control statement.

5.13 What type of repetition would be appropriate for calculating the sum of the first 100 positive integers? What type of repetition would be appropriate for calculating the sum of an arbitrary number of positive integers? Briefly describe how each of these tasks could be performed.

ANS: Counter-controlled repetition would be appropriate for calculating the sum of the first 100 positive integers, because the number of repetitions is known in advance. The application that performs this task could use a `while` repetition statement with a counter variable that takes on the values 1 to 100. The application could then add the current counter value to the total variable in each repetition of the loop. Sentinel-controlled repetition would be appropriate for calculating the sum of an arbitrary number of positive integers. The application that performs this task could use a sentinel value of `-1` to mark the end of data entry. The application could use a `while` repetition statement that totals positive integers from the user until the user enters the sentinel value.

5.14 What is the difference between the prefix increment operator and the postfix increment operator?

ANS: Incrementing a variable with the prefix increment operator causes the variable to be incremented by 1; then the new value of the variable is used in the expression in which it appears. Incrementing a variable with the postfix increment operator causes the current value of the variable to be used in the expression in which it appears; then the variable’s value is incremented by 1. The prefix increment and the postfix increment have the same effect when the incrementing operation appears in a statement by itself.

5.15 Identify and correct the errors in each of the following pieces of code. [Note: There may be more than one error in each piece of code.]

```
a) if ( age >= 65 );
    Console.WriteLine( "Age greater than or equal to 65" );
else
    Console.WriteLine( "Age is less than 65" );
```

ANS: The semicolon at the end of the `if` condition should be removed. The closing double quote of the second `Console.WriteLine` should be inside the closing parenthesis.

b) `int x = 1, total;`

```
while ( x <= 10 )
{
    total += x;
    ++x;
}
```

ANS: The variable `total` should be initialized to zero.

c) `while (x <= 100)`

```
total += x;
++x;
```

ANS: The two statements should be enclosed in curly braces to properly group them into the body of the `while`; otherwise the loop will be an infinite loop.

d) `while (y > 0)`

```
{
    Console.WriteLine( y );
    ++y;
```

ANS: The `++` operator should be changed to `--`; otherwise the loop will be an infinite loop if `y` is greater than 0 when the `while` statement is encountered. The closing curly brace for the `while` loop is missing.

5.16 What does the following application display?

```
1 // Exercise 5.16 Solution: Mystery.cs
2 using System;
3
4 public class Mystery
5 {
6     public static void Main( string[] args )
7     {
8         int y;
9         int x = 1;
10        int total = 0;
11
12        while ( x <= 10 )
13        {
14            y = x * x;
15            Console.WriteLine( y );
16            total += y;
17            ++x;
18        } // end while
19
20        Console.WriteLine( "Total is {0}", total );
21    } // end Main
22 } // end class Mystery
```

ANS:

```
1
4
9
16
25
36
49
64
81
100
Total is 385
```

For Exercise 5.17 through Exercise 5.20, perform each of the following steps:

- a) Read the problem statement.
- b) Formulate the algorithm using pseudocode and top-down, stepwise refinement.
- c) Write a C# application.
- d) Test, debug and execute the C# application.
- e) Process three complete sets of data.

5.17 Drivers are concerned with the mileage their automobiles get. One driver has kept track of several tankfuls of gasoline by recording the miles driven and gallons used for each tankful. Develop a C# application that will input the miles driven and gallons used (both as integers) for each tankful. The application should calculate and display the miles per gallon obtained for each tankful and display the combined miles per gallon obtained for all tankfuls up to this point. All averaging calculations should produce floating-point results. Display the results rounded to the nearest hundredth. Use the `Console` class's `ReadLine` method and sentinel-controlled repetition to obtain the data from the user.

ANS:

Top:

determine the current and combined miles/gallon for each tank of gas

First refinement:

initialize variables

input the miles driven and the gallons used

calculate and display the miles/gallon for each tank of gas

calculate and display the overall average miles/gallon

Second refinement:

initialize totalGallons to zero

initialize totalMiles to zero

prompt the user to enter the miles used for the first tank

input the miles used for the first tank (possibly the sentinel)

while the sentinel value (-1) has not been entered for the miles

prompt the user to enter the gallons used for the current tank

input the gallons used for the current tank

add miles to the running total in totalMiles

add gallons to the running total in totalGallons

if gallons is not zero

calculate and display the miles/gallon

if totalGallons is not zero

calculate and display the totalMiles/totalGallons

prompt the user for the next tank's number of miles

input the gallons used for the next tank

```

1  // Exercise 5.17 Solution: Gas.cs
2  // Application calculates average mpg
3  using System;
4
5  public class Gas
6  {
7      // perform miles-per-gallon calculations
8      public static void Main( string[] args )
9      {
10         int miles; // miles for one tankful
11         int gallons; // gallons for one tankful
12         int totalMiles = 0; // total miles for trip
13         int totalGallons = 0; // total gallons for trip
14
15         double milesPerGallon; // miles per gallon for tankful
16         double totalMilesPerGallon; // miles per gallon for trip
17
18         // prompt user for miles and obtain the input from user
19         Console.Write( "Enter miles (-1 to quit): " );
20         miles = Convert.ToInt32( Console.ReadLine() );
21
22         // exit if the input is -1; otherwise, proceed with the program
23         while ( miles != -1 )
24         {
25             // prompt user for gallons and obtain the input from user
26             Console.Write( "Enter gallons: " );
27             gallons = Convert.ToInt32( Console.ReadLine() );
28
29             // add gallons and miles for this tank to totals
30             totalMiles += miles;
31             totalGallons += gallons;

```

```
32
33     // calculate miles per gallon for the current tank
34     if ( gallons != 0 )
35     {
36         milesPerGallon = ( double ) miles / gallons;
37         Console.WriteLine( "MPG this tankful: {0:F}",
38             milesPerGallon );
39     } // end if statement
40
41     // calculate miles per gallon for the total trip
42     if ( totalGallons != 0 )
43     {
44         totalMilesPerGallon = ( double ) totalMiles / totalGallons;
45         Console.WriteLine( "Total MPG: {0:F}\n",
46             totalMilesPerGallon );
47     } // end if statement
48
49     // prompt user for new value for miles
50     Console.Write( "Enter miles (-1 to quit): " );
51     miles = Convert.ToInt32( Console.ReadLine() );
52     } // end while loop
53 } // end Main
54 } // end class Gas
```

```
Enter miles (-1 to quit): 100
Enter gallons: 4
MPG this tankful: 25.00
Total MPG: 25.00

Enter miles (-1 to quit): -1
```

```
Enter miles (-1 to quit): 450
Enter gallons: 14
MPG this tankful: 32.14
Total MPG: 32.14

Enter miles (-1 to quit): 200
Enter gallons: 6
MPG this tankful: 33.33
Total MPG: 32.50

Enter miles (-1 to quit): -1
```



```

Enter miles (-1 to quit): 425
Enter gallons: 14
MPG this tankful: 30.36
Total MPG: 30.36

Enter miles (-1 to quit): 336
Enter gallons: 12
MPG this tankful: 28.00
Total MPG: 29.27

Enter miles (-1 to quit): 405
Enter gallons: 15
MPG this tankful: 27.00
Total MPG: 28.44

Enter miles (-1 to quit): -1

```

5.18 Develop a C# application that will determine whether any of several department-store customers has exceeded the credit limit on a charge account. For each customer, the following facts are available:

- a) account number
- b) balance at the beginning of the month
- c) total of all items charged by the customer this month
- d) total of all credits applied to the customer's account this month
- e) allowed credit limit.

The application should input all of these facts as integers, calculate the new balance (= *beginning balance* + *charges* - *credits*), display the new balance and determine whether the new balance exceeds the customer's credit limit. For those customers whose credit limit is exceeded, the application should display the message "Credit limit exceeded". Use sentinel-controlled repetition to obtain the data for each account.

Top:

determine if each of an arbitrary number of department-store customers has exceeded the credit limit on a charge account

First refinement:

*input customer's data
 calculate and display the customer's new balance
 display message if user's balance exceeds credit limit
 process next customer*

Second refinement:

prompt the user for the first customer's account number

input the first customer's account number

while the sentinel value (-1) has not been entered for the account number

prompt the user for the customer's beginning balance

input the customer's beginning balance

prompt the user for the customer's total charges

input the customer's total charges

prompt the user for the customer's total credits

input the customer's total credits

prompt the user for the customer's credit limit

input the customer's credit limit

calculate and display the customer's new balance

if the balance exceeds the credit limit

display "Credit limit exceeded"

prompt the user for the next customer's account number

input the next customer's account number

```

1 // Exercise 5.18 Solution: Credit.cs
2 // Application monitors accounts.
3 using System;
4
5 public class Credit
6 {
7     // calculates the balance on several credit accounts
8     public static void Main( string[] args )
9     {
10         int account; // account number
11         int oldBalance; // starting balance
12         int charges; // total charges
13         int credits; // total credits
14         int creditLimit; // allowed credit limit
15         int newBalance; // new balance
16
17         Console.Write( "Enter Account Number (or -1 to quit): " );
18         // read in account number
19         account = Convert.ToInt32( Console.ReadLine() );
20
21         // exit if the input is -1; otherwise, continue
22         while ( account != -1 )
23         {
24             // prompt for and read original balance
25             Console.Write( "Enter Balance: " );
26             oldBalance = Convert.ToInt32( Console.ReadLine() );
27
28             // prompt for and read charges
29             Console.Write( "Enter Charges: " );
30             charges = Convert.ToInt32( Console.ReadLine() );

```

```

31
32     // prompt for and read credits
33     Console.Write( "Enter Credits: " );
34     credits = Convert.ToInt32( Console.ReadLine() );
35
36     // prompt for and read credit limit
37     Console.Write( "Enter Credit Limit: " );
38     creditLimit = Convert.ToInt32( Console.ReadLine() );
39
40     // calculate and display new balance
41     newBalance = oldBalance + charges - credits;
42     Console.WriteLine( "New balance is {0}", newBalance );
43
44     // display a warning if the user has exceed the credit limit
45     if ( newBalance > creditLimit )
46         Console.WriteLine( "Credit limit exceeded" );
47
48     // prompt for and read next account number
49     Console.Write( "\nEnter Account Number (or -1 to quit): " );
50     account = Convert.ToInt32( Console.ReadLine() );
51 } // end while
52 } // end Main
53 } // end class Credit

```

```

Enter Account Number (or -1 to quit): 10001
Enter Balance: 0
Enter Charges: 200
Enter Credits: 100
Enter Credit Limit: 500
New balance is 100

```

```

Enter Account Number (or -1 to quit): 10002
Enter Balance: 15000
Enter Charges: 30000
Enter Credits: 17
Enter Credit Limit: 20000
New balance is 44983
Credit limit exceeded

```

```

Enter Account Number (or -1 to quit): 10003
Enter Balance: 10
Enter Charges: 5
Enter Credits: 1
Enter Credit Limit: 100
New balance is 14

```

```

Enter Account Number (or -1 to quit): -1

```

5.19 A large company pays its salespeople on a commission basis. The salespeople receive \$200 per week plus 9% of their gross sales for that week. For example, a salesperson who sells \$5,000 worth of merchandise in a week receives \$200 plus 9% of \$5,000, or a total of \$650. You've been supplied with a list of items sold by each salesperson. The values of these items are as follows:

Item	Value
1	239.99
2	129.75
3	99.95
4	350.89

Develop a C# application that inputs one salesperson's items sold for last week and calculates and displays that salesperson's earnings. There is no limit to the number of items that can be sold by a salesperson.

ANS:

Top:

determine a salesperson's gross earnings for the week based on the sales commission from four products

First refinement:

*input the first product's sales for the week
add the gross earnings for that product to the total gross sales
process the next product of four
calculate and output the total earnings*

Second refinement:

*initialize gross sales to zero
initialize the product count to zero
while product count is less than four
 increment the product count
 input the number of items sold
 determine the cost per item based on the product count
 calculate the product's sales by multiplying the number of items sold
 by the cost per item
 add the product's sales to gross sales
calculate the salesperson's earnings as \$200 plus 9% of gross sales
output the salesperson's earnings*

```

1  // Exercise 5.19 Solution: Sales.cs
2  // Application calculates commissions based on sales.
3  using System;
4
5  public class Sales
6  {
7      // calculate sales for individual products
8      public static void Main( string[] args )
9      {
10         decimal grossSales = 0; // total gross sales
11         decimal earnings; // earnings made from sales
12         int product = 0; // the product number
13         int numberSold; // number sold of a given product
14
15         while ( product < 4 )
16         {

```

```

17         ++product;
18
19         // prompt for and read number of the product sold from user
20         Console.Write( "Enter number sold of product #{0}: ",
21             product );
22         numberSold = Convert.ToInt32( Console.ReadLine() );
23
24         // determine gross of each individual product and add to total
25         if ( product == 1 )
26             grossSales += numberSold * 239.99M;
27         else if ( product == 2 )
28             grossSales += numberSold * 129.75M;
29         else if ( product == 3 )
30             grossSales += numberSold * 99.95M;
31         else if ( product == 4 )
32             grossSales += numberSold * 350.89M;
33     } // end while loop
34
35     earnings = 0.09M * grossSales + 200; // calculate earnings
36     Console.WriteLine( "Earnings this week: {0:C}", earnings );
37 } // end Main
38 } // end class Sales

```

```

Enter number sold of product #1: 0
Enter number sold of product #2: 0
Enter number sold of product #3: 1
Enter number sold of product #4: 0
Earnings this week: $209.00

```

```

Enter number sold of product #1: 100
Enter number sold of product #2: 100
Enter number sold of product #3: 100
Enter number sold of product #4: 7
Earnings this week: $4,648.27

```

```

Enter number sold of product #1: 25
Enter number sold of product #2: 0
Enter number sold of product #3: 0
Enter number sold of product #4: 100
Earnings this week: $3,897.99

```

5.20 Develop a C# application that will determine the gross pay for each of three employees. The company pays straight time for the first 40 hours worked by each employee and time and a half for all hours worked in excess of 40 hours. You are given a list of the three employees of the company, the number of hours each employee worked last week and the hourly rate of each employee. Your application should input this information for each employee and should determine and display the employee's gross pay. Use the `Console` class's `ReadLine` method to input the data.

ANS:

Top:

determine the gross pay for three employees

First refinement:

initialize the variables

input the hours worked for the current worker

input the hourly rate of the current worker

calculate and display worker's gross pay

process the next worker of the three

Second refinement:

initialize count to 1

while the employee count is less than or equal to 3

prompt the user for the hours worked for the current employee

input the hours worked for the current employee

prompt the user for the employee's hourly rate

input the employee's hourly rate

if the hours input is less than or equal to 40

calculate gross pay by multiplying hours worked by hourly rate

else

calculate gross pay by multiplying 40 by the hourly rate, then

adding the product of the number of hours worked above 40

and 1.5 times the hourly rate

display the employee's gross pay

increment the employee count

```

1  // Exercise 5.20 Solution: Wages.cs
2  // Application calculates wages.
3  using System;
4
5  public class Wages
6  {
7      // calculates wages for 3 employees
8      public static void Main( string[] args )
9      {
10         decimal pay; // gross pay
11         int hours; // hours worked
12         decimal rate; // hourly rate
13         int count = 1; // number of employees
14
15         // repeat calculation for 3 employees
16         while ( count <= 3 )
17         {
18             // prompt user and read the hourly rate
19             Console.Write( "Enter hourly rate: " );
20             rate = Convert.ToDecimal( Console.ReadLine() );
21

```

```
22      // prompt user and read hours worked
23      Console.Write( "Enter hours worked: " );
24      hours = Convert.ToInt32( Console.ReadLine() );
25
26      // calculate wages
27      if ( hours <= 40 ) // straight time
28          pay = hours * rate;
29      else // with overtime
30          pay = ( 40 * rate ) + ( hours - 40 ) * ( rate * 1.5M );
31
32      // display the pay for the current employee
33      Console.WriteLine( "Pay for employee is {0:C}\n", pay );
34
35      ++count;
36      } // end while loop
37      } // end Main
38  } // end class Wages
```

Enter hourly rate: **7.25**
Enter hours worked: **50**
Pay for employee is \$398.75

Enter hourly rate: **8.50**
Enter hours worked: **40**
Pay for employee is \$340.00

Enter hourly rate: **10.00**
Enter hours worked: **30**
Pay for employee is \$300.00

Enter hourly rate: **15**
Enter hours worked: **0**
Pay for employee is \$0.00

Enter hourly rate: **6.50**
Enter hours worked: **40**
Pay for employee is \$260.00

Enter hourly rate: **300**
Enter hours worked: **60**
Pay for employee is \$21,000.00

Enter hourly rate: **18**
Enter hours worked: **20**
Pay for employee is \$360.00

Enter hourly rate: **12**
Enter hours worked: **30**
Pay for employee is \$360.00

Enter hourly rate: **9**
Enter hours worked: **40**
Pay for employee is \$360.00

5.21 The process of finding the maximum value (i.e., the largest of a group of values) is used frequently in computer applications. For example, an application that determines the winner of a sales contest would input the number of units sold by each salesperson. The salesperson who sells the most units wins the contest. Write a pseudocode application, then a C# application that inputs a series of 10 integers and determines and displays the largest integer. Your application should use at least the following three variables:

- a) counter: A counter to count to 10 (i.e., to keep track of how many numbers have been input and to determine when all 10 numbers have been processed).
- b) number: The integer most recently input by the user.
- c) largest: The largest number found so far.

ANS:

Pseudocode:

```

initialize counter to 0
prompt the user for the first number
input the first number
initialize largest with the first number
while the counter is less than 10
    prompt the user for the next number
    input the next number
    if the new number is greater than the largest
        make largest the new number
    increment the counter
  
```

```

1  // Exercise 5.21 Solution: Largest.cs
2  // Application determines and displays the largest of ten numbers.
3  using System;
4
5  public class Largest
6  {
7      // determine the largest of 10 numbers
8      public static void Main( string[] args )
9      {
10         int largest; // largest number
11         int number; // user input
12         int counter; // number of values entered
13
14         // get first number and assign it to variable largest
15         Console.Write( "Enter number: " );
16         largest = Convert.ToInt32( Console.ReadLine() );
17
18         counter = 1;
19
20         // get rest of the numbers and find the largest
21         while ( counter < 10 )
22         {
23             Console.Write( "Enter number: " );
24             number = Convert.ToInt32( Console.ReadLine() );
25
  
```

```

26         if ( number > largest )
27             largest = number;
28
29         ++counter;
30     } // end while loop
31
32     Console.WriteLine( "Largest number is {0}", largest );
33 } // end Main
34 } // end class Largest

```

```

Enter number: 100
Enter number: 300
Enter number: 200
Enter number: 3
Enter number: 4
Enter number: 6
Enter number: 7
Enter number: 8
Enter number: 4
Enter number: 3
Largest number is 300

```

5.22 Write a C# application that uses looping to display the following table of values:

N	10*N	100*N	1000*N
1	10	100	1000
2	20	200	2000
3	30	300	3000
4	40	400	4000
5	50	500	5000

ANS:

```

1  // Exercise 5.22 Solution: Table.cs
2  // Application displays a table of values using a while loop.
3  using System;
4
5  public class Table
6  {
7      public static void Main( string[] args )
8      {
9          int n = 1;
10
11         Console.WriteLine( "N\t10*N\t100*N\t1000*N\n" );
12
13         while ( n <= 5 )
14         {
15             Console.WriteLine( "{0}\t{1}\t{2}\t{3}",
16                                 n, ( 10 * n ), ( 100 * n ), ( 1000 * n ) );

```

```

17         ++n;
18     } // end while loop
19 } // end Main
20 } // end class Table

```

N	10*N	100*N	1000*N
1	10	100	1000
2	20	200	2000
3	30	300	3000
4	40	400	4000
5	50	500	5000

5.23 Using an approach similar to that for Exercise 5.21, find the *two* largest values of the 10 values entered. [Note: You may input each number only once.]

ANS:

```

1  // Exercise 5.23 Solution: TwoLargest.cs
2  // Application determines and displays the two largest of 10 numbers.
3  using System;
4
5  public class TwoLargest
6  {
7      // determines the two largest of 10 numbers
8      public static void Main( string[] args )
9      {
10         int largest; // largest number
11         int nextLargest; // second-largest number
12         int number; // user input
13         int counter; // number of values entered
14
15         // get first number and assign it to variable largest
16         Console.Write( "Enter number: " );
17         largest = Convert.ToInt32( Console.ReadLine() );
18
19         // get second number and compare it with first number
20         Console.Write( "Enter number: " );
21         number = Convert.ToInt32( Console.ReadLine() );
22
23         if ( number > largest )
24         {
25             nextLargest = largest;
26             largest = number;
27         } // end if
28         else
29             nextLargest = number;
30
31         counter = 2;
32
33         // get rest of the numbers and find the largest and nextLargest
34         while ( counter < 10 )
35         {

```

```

36     Console.Write( "Enter number: " );
37     number = Convert.ToInt32( Console.ReadLine() );
38
39     if ( number > largest )
40     {
41         nextLargest = largest;
42         largest = number;
43     } // end if
44     else if ( number > nextLargest )
45         nextLargest = number;
46
47     ++counter;
48 } // end while loop
49
50 Console.WriteLine( "Largest is {0}\nSecond largest is {1}",
51     largest, nextLargest );
52 } // end Main
53 } // end class TwoLargest

```

```

Enter number: 100
Enter number: 200
Enter number: 300
Enter number: 400
Enter number: 500
Enter number: 150
Enter number: 250
Enter number: 450
Enter number: 350
Enter number: 99999999
Largest is 99999999
Second largest is 500

```

5.24 Modify the application in Fig. 5.12 to validate its inputs. For any input, if the value entered is other than 1 or 2, display the message “Invalid input”, then keep looping until the user enters a correct value.

ANS:

```

1 // Exercise 5.24 Solution: Analysis.cs
2 // Application performs analysis of examination results.
3 using System;
4
5 public class Analysis
6 {
7     // analyze the results of 10 tests
8     public static void Main( string[] args )
9     {
10         // initializing variables in declarations
11         int passes = 0; // number of passes
12         int failures = 0; // number of failures
13         int studentCounter = 1; // student counter
14         int result; // one exam result
15

```

```

16      // process 10 students using counter-controlled loop
17      while ( studentCounter <= 10 )
18      {
19          // prompt user for input and obtain value from user
20          Console.Write( "Enter result (1 = pass, 2 = fail): " );
21          result = Convert.ToInt32( Console.ReadLine() );
22
23          // if...else nested in while
24          if ( result == 1 )
25          {
26              ++passes;
27              ++studentCounter;
28          } // end if
29          else if ( result == 2 )
30          {
31              ++failures;
32              ++studentCounter;
33          } // end else if
34          else
35              Console.WriteLine( "Invalid Input" );
36      } // end while
37
38      // termination phase; prepare and display results
39      Console.WriteLine( "Passed: {0}\nFailed: {1}", passes, failures );
40
41      // determine whether more than 8 students passed
42      if ( passes > 8 )
43          Console.WriteLine( "Bonus to instructor!" );
44  } // end Main
45 } // end class Analysis

```

```

Enter result (1 = pass, 2 = fail): 1
Enter result (1 = pass, 2 = fail): 1
Enter result (1 = pass, 2 = fail): 1
Enter result (1 = pass, 2 = fail): 3
Invalid Input
Enter result (1 = pass, 2 = fail): 2
Enter result (1 = pass, 2 = fail): 2
Enter result (1 = pass, 2 = fail): 0
Invalid Input
Enter result (1 = pass, 2 = fail): 1
Enter result (1 = pass, 2 = fail): 2
Enter result (1 = pass, 2 = fail): 1
Enter result (1 = pass, 2 = fail): 1
Enter result (1 = pass, 2 = fail): 1
Passed: 7
Failed: 3

```

5.25 What does the following application display?

```

1  // Exercise 5.25 Solution: Mystery2.cs
2  using System;
3

```

```

4 public class Mystery2
5 {
6     public static void Main( string[] args )
7     {
8         int count = 1;
9
10        while ( count <= 10 )
11        {
12            Console.WriteLine( count % 2 == 1 ? "****" : "+++++++" );
13            ++count;
14        } // end while
15    } // end Main
16 } // end class Mystery2

```

ANS:

```

****
+++++++
****
+++++++
****
+++++++
****
+++++++
****
+++++++

```

5.26 What does the following application display?

```

1 // Exercise 5.26 Solution: Mystery3.cs
2 using System;
3
4 public class Mystery3
5 {
6     public static void Main( string[] args )
7     {
8         int row = 10;
9         int column;
10
11        while ( row >= 1 )
12        {
13            column = 1;
14
15            while ( column <= 10 )
16            {
17                Console.Write( row % 2 == 1 ? "<" : ">" );
18                ++column;
19            } // end while
20
21            --row;

```

```

22         Console.WriteLine();
23     } // end while
24 } // end Main
25 } // end class Mystery3

```

ANS:

```

>>>>>>>>
<<<<<<<<<
>>>>>>>>
<<<<<<<<<
>>>>>>>>
<<<<<<<<<
>>>>>>>>
<<<<<<<<<
>>>>>>>>
<<<<<<<<<

```

5.27 (*Dangling-else Problem*) Determine the output for each of the given sets of code when x is 9 and y is 11 and when x is 11 and y is 9. Note that the compiler ignores the indentation in a C# application. Also, the C# compiler always associates an `else` with the immediately preceding `if` unless told to do otherwise by the placement of braces (`{}`). On first glance, you may not be sure which `if` an `else` matches—this situation is referred to as the “dangling-else problem.” We have eliminated the indentation from the following code to make the problem more challenging. [*Hint:* Apply the indentation conventions you’ve learned.]

```

a)  if ( x < 10 )
    if ( y > 10 )
        Console.WriteLine( "*****" );
    else
        Console.WriteLine( "#####" );
        Console.WriteLine( "$$$$$" );

```

ANS:

```

When:  x = 9, y = 11
*****
$$$$$

When:  x = 11, y = 9
$$$$$

```

```

b) if ( x < 10 )
{
    if ( y > 10 )
        Console.WriteLine( "*****" );
    }
    else
    {
        Console.WriteLine( "#####" );
        Console.WriteLine( "$$$$" );
    }
}

```

ANS:

When: x = 9, y = 11

When: x = 11, y = 9

\$\$\$\$

5.28 (*Another Dangling-else Problem*) Modify the given code to produce the output shown in each part of the problem. Use proper indentation techniques. Make no changes other than inserting braces and changing the indentation of the code. The compiler ignores indentation in a C# application. We have eliminated the indentation from the given code to make the problem more challenging. [Note: It is possible that no modification is necessary for some of the parts.]

```

if ( y == 8 )
if ( x == 5 )
    Console.WriteLine( "#####" );
else
    Console.WriteLine( "#####" );
    Console.WriteLine( "$$$$" );
    Console.WriteLine( "&&&&" );

```

a) Assuming that x = 5 and y = 8, the following output is produced:

```

####
$$$$
&&&&

```

ANS:

```

if ( y == 8 )
    if ( x == 5 )
        Console.WriteLine( "#####" );
    else
        Console.WriteLine( "#####" );
    Console.WriteLine( "$$$$" );
    Console.WriteLine( "&&&&" );

```

b) Assuming that $x = 5$ and $y = 8$, the following output is produced:

```
@@@@@
```

ANS:

```
if ( y == 8 )
    if ( x == 5 )
        Console.WriteLine( "@@@@@" );
    else
    {
        Console.WriteLine( "#####" );
        Console.WriteLine( "$$$$$" );
        Console.WriteLine( "&&&&&" );
    }
}
```

c) Assuming that $x = 5$ and $y = 8$, the following output is produced:

```
@@@@@
&&&&&
```

ANS:

```
if ( y == 8 )
    if ( x == 5 )
        Console.WriteLine( "@@@@@" );
    else
    {
        Console.WriteLine( "#####" );
        Console.WriteLine( "$$$$$" );
    }
Console.WriteLine( "&&&&&" );
```

d) Assuming that $x = 5$ and $y = 7$, the following output is produced.

```
#####
$$$$$
&&&&&
```

ANS:

```
if ( y == 8 )
    if ( x == 5 )
        Console.WriteLine( "@@@@@" );
    else
    {
        Console.WriteLine( "#####" );
        Console.WriteLine( "$$$$$" );
        Console.WriteLine( "&&&&&" );
    }
}
```

5.29 Write an application that prompts the user to enter the size of the side of a square, then displays a hollow square of that size made of asterisks. Your application should work for squares of all side lengths between 1 and 20. If the user enters a number less than 1 or greater than 20, your application should display a square of size 1 or 20, respectively.

ANS:

```
1 // Exercise 5.29 Solution: Hollow.cs
2 // Application displays a hollow square.
3 using System;
4
5 public class Hollow
6 {
```

```
7 // draw a hollow box surrounded by stars
8 public static void Main( string[] args )
9 {
10     int stars; // number of stars on a side
11     int column; // the current column of the square being displayed
12     int row = 1; // the current row of the square being displayed
13
14     // prompt and read the length of the side from the user
15     Console.Write( "Enter length of side: " );
16     stars = Convert.ToInt32( Console.ReadLine() );
17
18     if ( stars < 1 )
19     {
20         stars = 1;
21         Console.WriteLine( "Invalid Input\nUsing default value 1" );
22     } // end if
23     else if ( stars > 20 )
24     {
25         stars = 20;
26         Console.WriteLine( "Invalid Input\nUsing default value 20" );
27     } // end else if
28
29     // repeat for as many rows as the user entered
30     while ( row <= stars )
31     {
32         column = 1;
33
34         // and for as many columns as rows
35         while ( column <= stars )
36         {
37             if ( row == 1 )
38                 Console.Write( "*" );
39             else if ( row == stars )
40                 Console.Write( "*" );
41             else if ( column == 1 )
42                 Console.Write( "*" );
43             else if ( column == stars )
44                 Console.Write( "*" );
45             else
46                 Console.Write( " " );
47
48             ++column;
49         } // end inner while loop
50
51         Console.WriteLine();
52         ++row;
53     } // end outer while loop
54 } // end Main
55 } // end class Hollow
```

```
Enter length of side: 5
```

```
*****
*   *
*   *
*   *
*   *
*****
```

5.30 (*Palindromes*) A palindrome is a sequence of characters that reads the same backward as forward. For example, each of the following five-digit integers is a palindrome: 12321, 55555, 45554 and 11611. Write an application that reads in a five-digit integer and determines whether it is a palindrome. If the number is not five digits long, display an error message and allow the user to enter a new value. [*Hint*: Use the remainder and division operators to pick off the number's digits one at a time, from right to left.]

ANS:

```
1  // Exercise 5.30 Solution: Palindrome.cs
2  // Application tests for a palindrome
3  using System;
4
5  public class Palindrome
6  {
7      // checks if a 5-digit number is a palindrome
8      public static void Main( string[] args )
9      {
10         int number; // user input number
11         int digit1; // first digit
12         int digit2; // second digit
13         int digit4; // fourth digit
14         int digit5; // fifth digit
15         int digits; // number of digits in input
16
17         number = 0;
18         digits = 0;
19
20         // ask for a number until it is five digits
21         while ( digits != 5 )
22         {
23             Console.Write( "Enter a 5-digit number: " );
24             number = Convert.ToInt32( Console.ReadLine() );
25
26             if ( number < 100000 )
27             {
28                 if ( number > 9999 )
29                     digits = 5;
30                 else
31                     Console.WriteLine( "Number must be 5 digits" );
32             } // end if
33             else
34                 Console.WriteLine( "Number must be 5 digits" );
35         } // end while loop
36
```

```

37      // get the digits
38      digit1 = number / 10000;
39      digit2 = ( number % 10000 ) / 1000;
40      digit4 = ( number % 100 ) / 10;
41      digit5 = number % 10;
42
43      // display whether the number is a palindrome
44      Console.Write( number );
45
46      if ( digit1 == digit5 )
47      {
48          if ( digit2 == digit4 )
49              Console.WriteLine( " is a palindrome!!!" );
50          else
51              Console.WriteLine( " is not a palindrome." );
52      }
53      else
54          Console.WriteLine( " is not a palindrome." );
55      } // end Main
56 } // end class Palindrome

```

```

Enter a 5-digit number: 10000
10000 is not a palindrome.

```

```

Enter a 5-digit number: 12321
12321 is a palindrome!!!

```

```

Enter a 5-digit number: 999
Number must be 5 digits
Enter a 5-digit number: 12321
12321 is a palindrome!!!

```

5.31 Write an application that inputs an integer containing only 0s and 1s (i.e., a binary integer) and displays its decimal equivalent. [*Hint:* Picking the digits off a binary number is similar to picking the digits off a decimal number, which you did in Exercise 5.30. In the decimal number system, the rightmost digit has a positional value of 1 and the next digit to the left has a positional value of 10, then 100, then 1000 and so on. The decimal number 234 can be interpreted as $4 * 1 + 3 * 10 + 2 * 100$. In the binary number system, the rightmost digit has a positional value of 1, the next digit to the left has a positional value of 2, then 4, then 8 and so on. The decimal equivalent of binary 1101 is $1 * 1 + 0 * 2 + 1 * 4 + 1 * 8$, or $1 + 0 + 4 + 8$, or 13.]

ANS:

```

1  // Exercise 5.31 Solution: Binary.cs
2  // Application displays the decimal equivalent of a binary number.
3  using System;
4
5  public class Binary
6  {

```

```

7 // converts a binary number to decimal
8 public static void Main( string[] args )
9 {
10     int binary; // binary value
11     int binaryPositionalValue; // binary positional value
12     int decimalEquivalent; // decimal value
13
14     binaryPositionalValue = 1;
15     decimalEquivalent = 0;
16
17     // prompt for and read in a binary number
18     Console.Write( "Enter a binary number: " );
19     binary = Convert.ToInt32( Console.ReadLine() );
20
21     // convert to decimal equivalent
22     while ( binary != 0 )
23     {
24         decimalEquivalent += ( binary % 10 ) * binaryPositionalValue;
25         binary /= 10;
26         binaryPositionalValue *= 2;
27     } // end while loop
28
29     Console.WriteLine( "Decimal equivalent is: {0}",
30         decimalEquivalent );
31 } // end Main
32 } // end class Binary

```

```

Enter a binary number: 10001001
Decimal equivalent is: 137

```

5.32 Write an application that uses only the output statements

```

Console.Write( "*" );
Console.Write( " " );
Console.WriteLine();

```

to display the checkerboard pattern that follows. Note that a `Console.WriteLine` method call with no arguments causes the application to output a single newline character. [*Hint*: Repetition statements are required.]

```

* * * * *
 * * * * *
* * * * *
 * * * * *
* * * * *
 * * * * *
* * * * *
 * * * * *

```

ANS:

```

1 // Exercise 5.32 Solution: Stars.cs
2 // Application displays a checkerboard pattern.
3 using System;
4
5 public class Stars
6 {
7     public static void Main( string[] args )
8     {
9         int row = 1;
10
11         while ( row <= 8 )
12         {
13             int column = 1;
14
15             if ( row % 2 == 0 )
16                 Console.Write( " " );
17
18             while ( column <= 8 )
19             {
20                 Console.Write( "* " );
21                 ++column;
22             } // end inner while loop
23
24             Console.WriteLine();
25             ++row;
26         } // end outer while loop
27     } // end Main
28 } // end class Stars

```

```

* * * * *
 * * * * *
* * * * *
 * * * * *
* * * * *
 * * * * *
* * * * *
 * * * * *

```

5.33 Write an application that keeps displaying in the console window the powers of the integer 2—namely, 2, 4, 8, 16, 32, 64, and so on. Loop 40 times. What happens when you run this application?

ANS: The values quickly grow too large to be stored in an int variable.

```

1 // Exercise 5.33 Solution: Powers.cs
2 // Application displays powers of 2.
3 using System;
4
5 public class Powers
6 {

```

```
7 public static void Main( string[] args )
8 {
9     int x = 1;
10    int count = 0;
11
12    while ( count < 40 )
13    {
14        x *= 2;
15        ++count;
16        Console.WriteLine( x );
17    } // end while loop
18 } // end Main
19 } // end class Powers
```

```
2
4
8
16
32
64
128
256
512
1024
2048
4096
8192
16384
32768
65536
131072
262144
524288
1048576
2097152
4194304
8388608
16777216
33554432
67108864
134217728
268435456
536870912
1073741824
-2147483648
0
0
0
0
0
0
0
0
0
0
```

5.34 What is wrong with the following statement? Provide the correct statement to add one to the sum of *x* and *y*.

```
Console.WriteLine( ++(x + y) );
```

ANS: ++ can only be applied to a variable—not an expression with multiple terms. The correct statement is `Console.WriteLine(x + y + 1);`

5.35 (*Sides of a Triangle*) Write an application that reads three nonzero values entered by the user and determines and displays whether they could represent the sides of a triangle.

ANS:

```

1 // Exercise 5.35 Solution: Triangle1.cs
2 // Application takes three values and determines if
3 // they form the sides of a triangle.
4 using System;
5
6 public class Triangle1
7 {
8     // checks if three sides can form a triangle
9     public static void Main( string[] args )
10    {
11        double side1; // length of side 1
12        double side2; // length of side 2
13        double side3; // length of side 3
14        bool isTriangle; // whether the sides can form a triangle
15
16        // get values of three sides
17        Console.Write( "Enter side 1: " );
18        side1 = Convert.ToDouble( Console.ReadLine() );
19
20        Console.Write( "Enter side 2: " );
21        side2 = Convert.ToDouble( Console.ReadLine() );
22
23        Console.Write( "Enter side 3: " );
24        side3 = Convert.ToDouble( Console.ReadLine() );
25
26        // triangle testing
27        isTriangle = false;
28
29        if ( side1 + side2 > side3 )
30        {
31            if ( side2 + side3 > side1 )
32            {
33                if ( side3 + side1 > side2 )
34                    isTriangle = true;
35            } // end inner if statement
36        } // end outer if statement
37
38        if ( isTriangle )
39            Console.WriteLine( "These could be sides to a triangle " );
40        else
41            Console.WriteLine( "These do not form a triangle." );
42    } // end Main
43 } // end class Triangle1

```

```

Enter side 1: 3
Enter side 2: 4
Enter side 3: 5
These could be sides to a triangle

```

```

Enter side 1: 3
Enter side 2: 4
Enter side 3: 10
These do not form a triangle.

```

5.36 Write an application that reads three nonzero integers and determines and displays whether they could represent the sides of a right triangle.

ANS:

```

1  // Exercise 5.36 Solution: Triangle2.cs
2  // Application takes three integers and determines if they
3  // form the sides of a right triangle.
4  using System;
5
6  public class Triangle2
7  {
8      // checks if three sides can form a right triangle
9      public static void Main( string[] args )
10     {
11         int side1; // length of side 1
12         int side2; // length of side 2
13         int side3; // length of side 3
14         bool isRightTriangle; // whether the sides can form a triangle
15
16         // get values of three sides
17         Console.Write( "Enter side 1: " );
18         side1 = Convert.ToInt32( Console.ReadLine() );
19
20         Console.Write( "Enter side 2: " );
21         side2 = Convert.ToInt32( Console.ReadLine() );
22
23         Console.Write( "Enter side 3: " );
24         side3 = Convert.ToInt32( Console.ReadLine() );
25
26         // square the sides
27         int side1Square = side1 * side1;
28         int side2Square = side2 * side2;
29         int side3Square = side3 * side3;
30
31         // test if these form a right triangle
32         isRightTriangle = false;
33
34         if ( ( side1Square + side2Square ) == side3Square )
35             isRightTriangle = true;
36         else if ( ( side1Square + side3Square ) == side2Square )
37             isRightTriangle = true;

```

```

38         else if ( ( side2Square + side3Square ) == side1Square )
39             isRightTriangle = true;
40
41         if ( isRightTriangle )
42             Console.WriteLine( "These are the sides of a right triangle." );
43         else
44             Console.WriteLine( "These do not form a right triangle." );
45     } // end Main
46 } // end class Triangle2

```

```

Enter side 1: 3
Enter side 2: 4
Enter side 3: 5
These are the sides of a right triangle.

```

```

Enter side 1: 3
Enter side 2: 4
Enter side 3: 6
These do not form a right triangle.

```

5.37 (*Factorials*) The factorial of a nonnegative integer n is written as $n!$ (pronounced “ n factorial”) and is defined as follows:

$$n! = n \cdot (n-1) \cdot (n-2) \cdot \dots \cdot 1 \quad (\text{for values of } n \text{ greater than or equal to } 1)$$

and

$$n! = 1 \quad (\text{for } n = 0)$$

For example, $5! = 5 \cdot 4 \cdot 3 \cdot 2 \cdot 1$, which is 120. Write an application that reads a nonnegative integer and computes and displays its factorial.

ANS:

```

1  // Exercise 5.37 Solution: Factorial.cs
2  // Application calculates a factorial.
3  using System;
4
5  public class Factorial
6  {
7      // calculates the factorial of a number
8      public static void Main( string[] args )
9      {
10         int number; // user input
11         int factorial; // factorial of input value
12
13         factorial = 1;
14
15         Console.Write( "Enter a positive integer: " );
16         number = Convert.ToInt32( Console.ReadLine() );
17
18         Console.Write( "{0}! is ", number );
19

```

```

20      // calculate factorial
21      while ( number > 0 )
22      {
23          factorial *= number;
24          --number;
25      } // end while loop
26
27      Console.WriteLine( factorial );
28  } // end Main
29 } // end class Factorial

```

Enter a positive integer: 7
7! is 5040

5.38 (*Infinite Series: Mathematical Constant e*) Write an application that estimates the value of the mathematical constant e by using the formula

$$e = 1 + \frac{1}{1!} + \frac{1}{2!} + \frac{1}{3!} + \dots$$

Note that the predefined constant `Math.E` (class `Math` is in the `System` namespace) provides a good approximation of e . Use the `WriteLine` method to output both your estimated value of e and `Math.E` for comparison.

ANS:

```

1  // Exercise 5.38 Solution: E.cs
2  // Application calculates estimated value of e.
3  using System;
4
5  public class E
6  {
7      // approximates the value of e
8      public static void Main( string[] args )
9      {
10         int number; // counter
11         int accuracy; // number of terms in the estimate
12         double factorial; // value of the factorial
13         double e; // estimated value of e
14
15         number = 1;
16         factorial = 1;
17         e = 1.0;
18
19         Console.Write( "Enter desired accuracy of e: " );
20         accuracy = Convert.ToInt32( Console.ReadLine() );
21
22         // calculate estimation
23         while ( number < accuracy )
24         {
25             factorial *= number;
26             e += 1.0 / factorial;
27             ++number;
28         } // end while loop

```

```

29
30     Console.Write( "estimated e is " );
31     Console.WriteLine( e );
32
33     Console.WriteLine( "Math.E is {0}", Math.E );
34 } // end Main
35 } // end class E

```

```

Enter desired accuracy of e: 3
estimated e is 2.5
Math.E is 2.71828182845905

```

```

Enter desired accuracy of e: 10
estimated e is 2.71828152557319
Math.E is 2.71828182845905

```

```

Enter desired accuracy of e: 40
estimated e is 2.71828182845905
Math.E is 2.71828182845905

```

5.39 (*Infinite Series: e^x*) Write an application that computes the value of e^x by using the formula

$$e^x = 1 + \frac{x}{1!} + \frac{x^2}{2!} + \frac{x^3}{3!} + \dots$$

Compare the result of your calculation to the return value of the method call

```
Math.Pow( Math.E, x )
```

[*Note:* The predefined method `Math.Pow` takes two arguments and raises the first argument to the power of the second. We discuss `Math.Pow` in Section 6.4.]

ANS:

```

1  // Exercise 5.39 Solution: EtoX.cs
2  // Application calculates e raised to x.
3  using System;
4
5  public class EtoX
6  {
7      // approximates the value of e to the x
8      public static void Main( string[] args )
9      {
10         int number; // counter
11         int accuracy; // accuracy of estimate
12         long factorial; // value of factorial
13         int x; // x value
14         double e; // estimated value of e
15         double exponent; // exponent value
16
17         number = 1;

```

```

18 factorial = 1;
19 e = 1.0;
20 exponent = 1.0;
21
22 Console.Write( "Enter exponent: " );
23 x = Convert.ToInt32( Console.ReadLine() );
24
25 Console.Write( "Enter desired accuracy of e: " );
26 accuracy = Convert.ToInt32( Console.ReadLine() );
27
28 // calculate estimation
29 while ( number < accuracy )
30 {
31     exponent *= x;
32     factorial *= number;
33     e += exponent / factorial;
34     ++number;
35 } // end while loop
36
37 Console.WriteLine( "estimated e to the {0} is {1}", x, e );
38
39 Console.WriteLine( "Math.E to the {0} is {1}",
40     x, Math.Pow( Math.E, x ) );
41 } // end Main
42 } // end class EtoX

```

```

Enter exponent: 3
Enter desired accuracy of e: 3
estimated e to the 3 is 8.5
Math.E to the 3 is 20.0855369231877

```

```

Enter exponent: 3
Enter desired accuracy of e: 10
estimated e to the 3 is 20.0633928571429
Math.E to the 3 is 20.0855369231877

```

```

Enter exponent: 3
Enter desired accuracy of e: 20
estimated e to the 3 is 20.0855369215177
Math.E to the 3 is 20.0855369231877

```

Making a Difference Exercises

5.40 (*World Population Growth*) World population has grown considerably over the centuries. Continued growth could eventually challenge the limits of breathable air, drinkable water, arable cropland and other limited resources. There is evidence that growth has been slowing in recent years and that world population could peak some time this century, then start to decline.

For this exercise, research world population growth issues online. *Be sure to investigate various viewpoints.* Get estimates for the current world population and its growth rate (the percentage by which it is likely to increase this year). Write a program that calculates world population growth

each year for the next 75 years, *using the simplifying assumption that the current growth rate will stay constant*. When displaying the results, the first column should display the year from year 1 to year 75. The second column should display the anticipated world population at the end of that year. The third column should display the numerical increase in the world population that would occur that year. Using your results, determine the year in which the population would be double what it is today, if this year's growth rate were to persist. [Hint: Use double variables because int variables can store values only up to approximately two billion. Display the double values using the format F0.]

ANS:

```

1  // Exercise 5.40 Solution: WorldPopulationGrowth.cs
2  // Based on the current annual population growth rate and current
3  // world population, calculate the world population.
4  using System;
5
6  public class WorldPopulationGrowth
7  {
8      public static void Main( string[] args )
9      {
10         long currentPopulation; // today's world population
11         double growthRate; // growth rate (1.14% would be .0114)
12         double futurePopulation; // future population based on growth rate
13
14         Console.WriteLine( "Welcome to the world population calculator" );
15         Console.Write( "Enter the current world population: " );
16         currentPopulation = Convert.ToInt64( Console.ReadLine() );
17         Console.Write(
18             "Enter the current growth rate: (e.g, 1.14% would be .0114): " );
19         growthRate = Convert.ToDouble( Console.ReadLine() );
20
21         int year = 1; // year counter
22         Console.WriteLine(
23             "Year\tEstimated Population\tChange from prior Year" );
24
25         while ( year <= 75 )
26         {
27             futurePopulation = currentPopulation * ( 1 + growthRate );
28             Console.WriteLine( "{0}\t{1}\t\t{2}",
29                 year, (long) futurePopulation,
30                 (long) futurePopulation - currentPopulation );
31             currentPopulation = (long) futurePopulation; // keep track
32             ++year;
33         } // end while
34     } // end Main
35 } // end class WorldPopulationGrowth

```

Welcome to the world population calculator
 Enter the current world population: **6829510656**
 Enter the current growth rate: (e.g, 1.14% would be .0114): **.0117**

Year	Estimated Population	Change from prior Year
1	6909415930	79905274
2	6990256096	80840166
3	7072042092	81785996
4	7154784984	82742892
5	7238495968	83710984
6	7323186370	84690402
7	7408867650	85681280
8	7495551401	86683751
9	7583249352	87697951
10	7671973369	88724017
11	7761735457	89762088
12	7852547761	90812304
13	7944422569	91874808
14	8037372313	92949744
15	8131409569	94037256
16	8226547060	95137491
17	8322797660	96250600
18	8420174392	97376732
19	8518690432	98516040
20	8618359110	99668678
21	8719193911	100834801
22	8821208479	102014568
23	8924416618	103208139
24	9028832292	104415674
25	9134469629	105637337
26	9241342923	106873294
27	9349466635	108123712
28	9458855394	109388759
29	9569524002	110668608
30	9681487432	111963430
31	9794760834	113273402
32	9909359535	114598701
33	10025299041	115939506
34	10142595039	117295998
35	10261263400	118668361
36	10381320181	120056781
37	10502781627	121461446
38	10625664172	122882545
39	10749984442	124320270
40	10875759259	125774817
41	11003005642	127246383
42	11131740808	128735166
43	11261982175	130241367
44	11393747366	131765191
45	11527054210	133306844
46	11661920744	134866534
47	11798365216	136444472
48	11936406089	138040873
49	12076062040	139655951
50	12217351965	141289925

50	12217351965	141289925
51	12360294982	142943017
52	12504910433	144615451
53	12651217885	146307452
54	12799237134	148019249
55	12948988208	149751074
56	13100491370	151503162
57	13253767119	153275749
58	13408836194	155069075
59	13565719577	156883383
60	13724438496	158718919
61	13885014426	160575930
62	14047469094	162454668
63	14211824482	164355388
64	14378102828	166278346
65	14546326631	168223803
66	14716518652	170192021
67	14888701920	172183268
68	15062899732	174197812
69	15239135658	176235926
70	15417433545	178297887
71	15597817517	180383972
72	15780311981	182494464
73	15964941631	184629650
74	16151731448	186789817
75	16340706705	188975257

5.41 (*Enforcing Privacy with Cryptography*) The explosive growth of Internet communications and data storage on Internet-connected computers has greatly increased privacy concerns. The field of cryptography is concerned with coding data to make it difficult (and hopefully—with the most advanced schemes—impossible) for unauthorized users to read. In this exercise you'll investigate a simple scheme for encrypting and decrypting data. A company that wants to send data over the Internet has asked you to write a program that will encrypt it so that it may be transmitted more securely. All the data is transmitted as four-digit integers. Your application should read a four-digit integer entered by the user and encrypt it as follows: Replace each digit with the result of adding 7 to the digit and getting the remainder after dividing the new value by 10. Then swap the first digit with the third, and swap the second digit with the fourth. Then display the encrypted integer. Write a separate application that inputs an encrypted four-digit integer and decrypts it (by reversing the encryption scheme) to form the original number. Use the format specifier D4 to display the encrypted value in case the number starts with a 0.

ANS:

```

1 // Exercise 5.41 Part A Solution: Encrypt.cs
2 // Application encrypts a four-digit number.
3 using System;
4
5 public class Encrypt
6 {
7     // encrypt a four-digit number
8     public static void Main( string[] args )
9     {
10         int number; // original number

```

```

11     int digit1; // first digit
12     int digit2; // second digit
13     int digit3; // third digit
14     int digit4; // fourth digit
15     int encryptedNumber; // encrypted number
16
17     // enter four-digit number to be encrypted
18     Console.Write( "Enter a four-digit number: " );
19     number = Convert.ToInt32( Console.ReadLine() );
20
21     // encrypt
22     digit1 = ( ( number / 1000 ) + 7 ) % 10;
23     digit2 = ( ( ( number % 1000 ) / 100 ) + 7 ) % 10;
24     digit3 = ( ( ( number % 100 ) / 10 ) + 7 ) % 10;
25     digit4 = ( ( number % 10 ) + 7 ) % 10;
26
27     encryptedNumber = digit1 * 10 + digit2 +
28         digit3 * 1000 + digit4 * 100;
29
30     Console.WriteLine( "Encrypted number is {0:D4}", encryptedNumber );
31 } // end Main
32 } // end class Encrypt

```

Enter a four-digit number: 1234
 Encrypted number is 0189

```

1 // Exercise 5.41 Part B Solution: Decrypt.cs
2 // Application decrypts a four-digit number.
3 using System;
4
5 public class Decrypt
6 {
7     // decrypt a four-digit number
8     public static void Main( string[] args )
9     {
10         int number; // original number
11         int digit1; // first digit
12         int digit2; // second digit
13         int digit3; // third digit
14         int digit4; // fourth digit
15         int decryptedNumber; // encrypted number
16
17         // enter four-digit number to be decrypted
18         Console.Write( "Enter a four-digit number: " );
19         number = Convert.ToInt32( Console.ReadLine() );
20
21         // decrypt
22         digit1 = ( ( number / 1000 ) + 3 ) % 10;
23         digit2 = ( ( ( number % 1000 ) / 100 ) + 3 ) % 10;
24         digit3 = ( ( ( number % 100 ) / 10 ) + 3 ) % 10;
25         digit4 = ( ( number % 10 ) + 3 ) % 10;
26

```


```
27         decryptedNumber = digit1 * 10 + digit2 +  
28             digit3 * 1000 + digit4 * 100;  
29  
30         Console.WriteLine( "Decrypted number is {0}", decryptedNumber );  
31     } // end Main  
32 } // end class Decrypt
```

Enter a four-digit number: **0189**
Decrypted number is 1234

Control Statements: Part 2

Solutions

6



Not everything that can be counted counts, and not everything that counts can be counted.

—Albert Einstein

Who can control his fate?

—William Shakespeare

The used key is always bright.

—Benjamin Franklin

Intelligence ... is the faculty of making artificial objects, especially tools to make tools.

—Henri Bergson

Every advantage in the past is judged in the light of the final issue.

—Demosthenes

Objectives

In this chapter you'll learn:

- The essentials of counter-controlled repetition.
- To use the **for** and **do...while** repetition statements.
- To use the **switch** multiple selection statement.
- To use the **break** and **continue** statements to alter the flow of control.
- To use the logical operators to form complex conditional expressions.

Self-Review Exercises

6.1 Fill in the blanks in each of the following statements:

- a) Typically, _____ statements are used for counter-controlled repetition and _____ statements are used for sentinel-controlled repetition.

ANS: for, while.

- b) The `do...while` statement tests the loop-continuation condition _____, executing the loop's body; therefore, the body always executes at least once.

ANS: after.

- c) The _____ statement selects among multiple actions based on the possible values of an integer variable or expression.

ANS: switch.

- d) The _____ statement, when executed in a repetition statement, skips the remaining statements in the loop body and proceeds with the next iteration of the loop.

ANS: continue.

- e) The _____ operator can be used to ensure that two conditions are *both* true before choosing a certain path of execution.

ANS: `&&` (conditional AND) or `&` (boolean logical AND).

- f) If the loop-continuation condition in a `for` header is initially _____, the `for` statement's body does not execute.

ANS: false.

- g) Methods that perform common tasks and do not need to be called on objects are called _____ methods.

ANS: static.

6.2 State whether each of the following is *true* or *false*. If *false*, explain why.

- a) The default label is required in the `switch` selection statement.

ANS: False. The default label is optional. If no default action is needed, then there is no need for a default label.

- b) The `break` statement is required in every case of a `switch` statement.

ANS: False. You could terminate the case with other statements, such as a `return`.

- c) The expression `((x > y) && (a < b))` is true if either `(x > y)` is true or `(a < b)` is true.

ANS: False. Both of the relational expressions must be true for this entire expression to be true when using the `&&` operator.

- d) An expression containing the `||` operator is true if either or both of its operands are true.

ANS: True.

- e) The integer after the comma (,) in a format item (e.g., `{0,4}`) indicates the field width of the displayed string.

ANS: True.

- f) To test for a range of values in a `switch` statement, use a hyphen (-) between the start and end values of the range in a case label.

ANS: False. The `switch` statement does not provide a mechanism for testing ranges of values, so you must list every value to test in a separate case label.

- g) Listing cases consecutively with no statements between them enables the cases to perform the same set of statements.

ANS: True.

6.3 Write a C# statement or a set of C# statements to accomplish each of the following tasks:

- a) Sum the odd integers between 1 and 99, using a `for` statement. Assume that the integer variables `sum` and `count` have been declared.

ANS:

```

1 // Exercise 6.3a Solution: PartA.cs
2 using System;
3
4 public class PartA
5 {
6     public static void Main( string[] args )
7     {
8         int sum = 0;
9         int count;
10
11         for ( count = 1; count <= 99; count += 2 )
12             sum += count;
13
14         Console.WriteLine( "sum = {0}", sum );
15     } // end Main
16 } // end class PartA

```

sum = 2500

b) Calculate the value of 2.5 raised to the power of 3, using the Pow method.

ANS:

```

1 // Exercise 6.3b Solution: PartB.cs
2 using System;
3
4 public class PartB
5 {
6     public static void Main( string[] args )
7     {
8         double result = Math.Pow( 2.5, 3 );
9
10        Console.WriteLine( "2.5 raised to the power of 3 is {0:F3}",
11                           result );
12    } // end Main
13 } // end class PartB

```

2.5 raised to the power of 3 is 15.625

c) Display the integers from 1 to 20 using a `while` loop and the counter variable `i`. Assume that the variable `i` has been declared, but not initialized. Display only five integers per line. [Hint: Use the calculation `i % 5`. When the value of this expression is 0, display a newline character; otherwise, display a tab character. Use the `Console.WriteLine()` method to output the newline character, and use the `Console.Write('\t')` method to output the tab character.]

4 Chapter 6 Control Statements: Part 2 Solutions

ANS:

```
1 // Exercise 6.3c Solution: PartC.cs
2 using System;
3
4 public class PartC
5 {
6     public static void Main( string[] args )
7     {
8         int i = 1;
9
10        while ( i <= 20 )
11        {
12            Console.Write( i );
13
14            if ( i % 5 == 0 )
15                Console.WriteLine();
16            else
17                Console.Write( '\t' );
18
19            ++i;
20        } // end while
21    } // end Main
22 } // end class PartC
```

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

d) Repeat part (c), using a for statement.

ANS:

```
1 // Exercise 6.3d Solution: PartD.cs
2 using System;
3
4 public class PartD
5 {
6     public static void Main( string[] args )
7     {
8         for ( int i = 1; i <= 20; i++ )
9         {
10            Console.Write( i );
11
12            if ( i % 5 == 0 )
13                Console.WriteLine();
14            else
15                Console.Write( '\t' );
16        } // end for
17    } // end Main
18 } // end class PartD
```

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

6.4 Find the error in each of the following code segments and explain how to correct it:

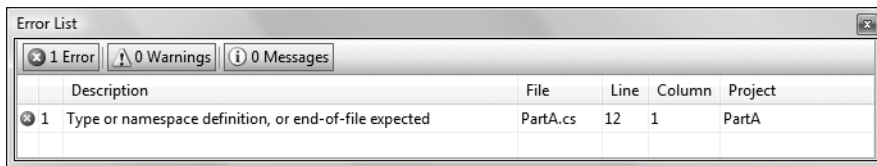
a) `i = 1;`

```
while ( i <= 10 );
    ++i;
}
```

ANS: Error: The semicolon after the `while` header causes an infinite loop, and there is a missing left brace for the body of the `while` statement.

Correction: Remove the semicolon and add a `{` before the loop's body.

```
1 // Exercise 6.4a: PartA.cs
2 public class PartA
3 {
4     public static void Main( string[] args )
5     {
6         int i = 1;
7
8         while ( i <= 10 );
9             ++i;
10    } // end while
11 } // end Main
12 } // end class PartA
```



```
1 // Exercise 6.4a Solution: PartACorrected.cs
2 public class PartACorrected
3 {
4     public static void Main( string[] args )
5     {
6         int i = 1;
7
8         while ( i <= 10 )
9         {
10            ++i;
11        } // end while
12    } // end Main
13 } // end class PartACorrected
```

6 Chapter 6 Control Statements: Part 2 Solutions

b) **for** (**k** = 0.1; **k** != 1.0; **k** += 0.1)
 Console.WriteLine(**k**);

ANS: Error: Using a floating-point number to control a **for** statement may not work, because floating-point numbers are represented only approximately by most computers.

Correction: Use an integer, and perform the proper calculation in order to get the values you desire. Also, the continuation condition of a **for** loop usually uses a relational operator (such as <) instead of an equality operator

```
1 // Exercise 6.4b: PartB.cs
2 using System;
3
4 public class PartB
5 {
6     public static void Main( string[] args )
7     {
8         double k;
9
10        for ( k = 0.1; k != 1.0; k += 0.1 )
11            Console.WriteLine( k );
12    } // end Main
13 } // end class PartB
```

```
0.1
0.2
0.3
0.4
0.5
0.6
0.7
0.8
0.9
1
1.1
1.2
1.3
1.4
1.5
...
```

```
1 // Exercise 6.4b Solution: PartBCorrected.cs
2 using System;
3
4 public class PartBCorrected
5 {
6     public static void Main( string[] args )
7     {
8         int k;
9
10        for ( k = 1; k < 10; k++ )
11            Console.WriteLine( ( double ) k / 10 );
12    } // end Main
13 } // end class PartBCorrected
```

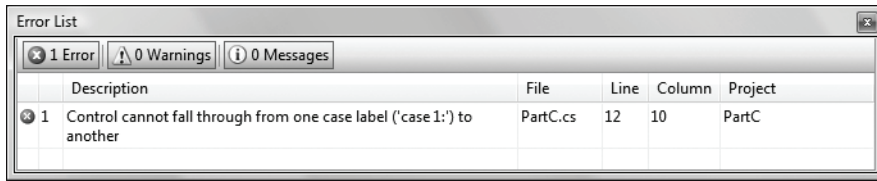
```
0.1
0.2
0.3
0.4
0.5
0.6
0.7
0.8
0.9
```

```
c) switch ( n )
{
    case 1:
        Console.WriteLine( "The number is 1" );
    case 2:
        Console.WriteLine( "The number is 2" );
        break;
    default:
        Console.WriteLine( "The number is not 1 or 2" );
        break;
}
```

ANS: Error: case 1 cannot fall through into case 2.

Correction: Terminate the case in some way, such as adding a break statement at the end of the statements for the first case.

```
1  // Exercise 6.4c: PartC.cs
2  using System;
3
4  public class PartC
5  {
6      public static void Main( string[] args )
7      {
8          int n = 1;
9
10         switch ( n )
11         {
12             case 1:
13                 Console.WriteLine( "The number is 1" );
14             case 2:
15                 Console.WriteLine( "The number is 2" );
16                 break;
17             default:
18                 Console.WriteLine( "The number is not 1 or 2" );
19                 break;
20         } // end switch
21     } // end Main
22 } // end class PartC
```



```

1 // Exercise 6.4c: PartCCorrected.cs
2 using System;
3
4 public class PartCCorrected
5 {
6     public static void Main( string[] args )
7     {
8         int n = 1;
9
10        switch ( n )
11        {
12            case 1:
13                Console.WriteLine( "The number is 1" );
14                break;
15            case 2:
16                Console.WriteLine( "The number is 2" );
17                break;
18            default:
19                Console.WriteLine( "The number is not 1 or 2" );
20                break;
21        } // end switch
22    } // end Main
23 } // end class PartCCorrected

```

The number is 1

d) The following code should display the values 1 to 10:

```
n = 1;
```

```

while ( n < 10 )
    Console.WriteLine( n++ );

```

ANS: Error: An improper relational operator is used in the while condition.

Correction: Use <= rather than <, or change 10 to 11.

```

1 // Exercise 6.4d: PartD.cs
2 using System;
3
4 public class PartD
5 {
6     public static void Main( string[] args )
7     {

```

```
8      int n = 1;
9
10     while ( n < 10 )
11         Console.WriteLine( n++ );
12     } // end Main
13 } // end class PartD
```

```
1
2
3
4
5
6
7
8
9
```

```
1 // Exercise 6.4d: PartDCorrected.cs
2 using System;
3
4 public class PartDCorrected
5 {
6     public static void Main( string[] args )
7     {
8         int n = 1;
9
10        while ( n <= 10 )
11            Console.WriteLine( n++ );
12    } // end Main
13 } // end class PartDCorrected
```

```
1
2
3
4
5
6
7
8
9
10
```

Exercises

- 6.5** Describe the four basic elements of counter-controlled repetition.

ANS: Counter-controlled repetition requires a control variable (or loop counter), an initial value of the control variable, an increment (or decrement) by which the control variable is modified each time through the loop, and a loop-continuation condition that determines whether looping should continue.

- 6.6** Compare and contrast the `while` and `for` repetition statements.

ANS: The `while` and `for` repetition statements repeatedly execute a statement or set of statements as long as a loop-continuation condition remains true. Both statements execute their bodies zero or more times. The `for` repetition statement specifies the counter-controlled-repetition details in its header, whereas the control variable in a `while` statement normally is initialized before the loop and incremented in the loop's body. Typically, `for` statements are used for counter-controlled repetition, and `while` statements for sentinel-controlled repetition. However, `while` and `for` can each be used for either repetition type.

- 6.7** Discuss a situation in which it would be more appropriate to use a `do...while` statement than a `while` statement. Explain why.

ANS: If you want a statement or set of statements to execute at least once, then repeat based on a condition, a `do...while` is more appropriate than a `while` (or a `for`). A `do...while` statement tests the loop-continuation condition *after* executing the loop's body; therefore, the body always executes at least once. A `while` tests the loop-continuation condition before executing the loop's body, so the application would need to include the statement(s) required to execute at least once both before the loop and in the body of the loop. Using a `do...while` avoids this duplication of code. Suppose an application needs to obtain an integer value from the user, and the integer value entered must be positive for the application to continue. In this case, a `do...while`'s body could contain the statements required to obtain the user input, and the loop-continuation condition could determine whether the value entered is less than 0. If so, the loop would repeat and prompt the user for input again. This would continue until the user entered a value greater than or equal to zero. Once this criterion was met, the loop-continuation condition would become false, and the loop would terminate, allowing the application to continue past the loop. This process is often called validating input.

- 6.8** Compare and contrast the `break` and `continue` statements.

ANS: The `break` and `continue` statements alter the flow of control through a control statement. The `break` statement, when executed in one of the repetition statements, causes immediate exit from that statement. Execution typically continues with the first statement after the control statement. In contrast, the `continue` statement, when executed in a repetition statement, skips the remaining statements in the loop body and proceeds with the next iteration of the loop. In `while` and `do...while` statements, the application evaluates the loop-continuation test immediately after the `continue` statement executes. In a `for` statement, the increment expression executes, then the application evaluates the loop-continuation test.

- 6.9** Find and correct the error(s) in each of the following segments of code:

a)

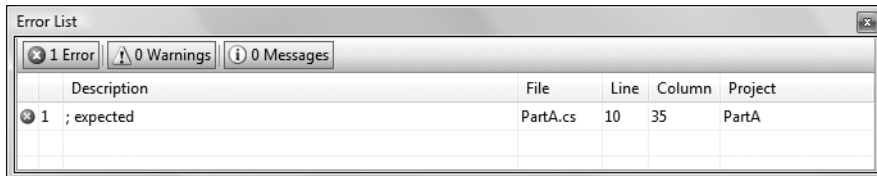
```
For ( i = 100, i >= 1, i++ )
    Console.WriteLine( i );
```

ANS: The F in for should be lowercase. Semicolons should be used in the for header instead of commas. The operator ++ should be --.

```

1 // Exercise 6.9a: PartA.cs
2 using System;
3
4 public class PartA
5 {
6     public static void Main( string[] args )
7     {
8         int i;
9
10        For ( i = 100, i >= 1, i++ )
11            Console.WriteLine( i );
12    } // end Main
13 } // end class PartA

```



```

1 // Exercise 6.9a Solution: PartACorrected.cs
2 using System;
3
4 public class PartACorrected
5 {
6     public static void Main( string[] args )
7     {
8         int i;
9
10        for ( i = 100; i >= 1; i-- )
11            Console.WriteLine( i );
12    } // end Main
13 } // end class PartACorrected

```

```

100
99
98
97
...
3
2
1

```

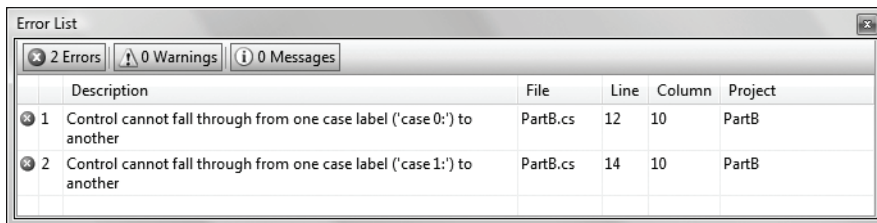
12 Chapter 6 Control Statements: Part 2 Solutions

b) The following code should display whether integer value is odd or even:

```
switch ( value % 2 )
{
    case 0:
        Console.WriteLine( "Even integer" );
    case 1:
        Console.WriteLine( "Odd integer" );
}
```

ANS: A break statement should be placed at the end of the statements in case 0: and case 1:.

```
1 // Exercise 6.9b: PartB.cs
2 using System;
3
4 public class PartB
5 {
6     public static void Main( string[] args )
7     {
8         int value = 8;
9
10        switch ( value % 2 )
11        {
12            case 0:
13                Console.WriteLine( "Even integer" );
14            case 1:
15                Console.WriteLine( "Odd integer" );
16        } // end switch
17    } // end Main
18 } // end class PartB
```



```
1 // Exercise 6.9b Solution: PartBCorrected.cs
2 using System;
3
4 public class PartBCorrected
5 {
6     public static void Main( string[] args )
7     {
8         int value = 8;
9
10        switch ( value % 2 )
11        {
```

```

12         case 0:
13             Console.WriteLine( "Even integer" );
14             break;
15         case 1:
16             Console.WriteLine( "Odd integer" );
17             break;
18     } // end switch
19 } // end Main
20 } // end class PartBCorrected

```

Even integer

c) The following code should output the odd integers from 19 to 1:

```

for ( int i = 19; i >= 1; i += 2 )
    Console.WriteLine( i );

```

ANS: The += operator in the for header should be -=.

```

1 // Exercise 6.9c: PartC.cs
2 using System;
3
4 public class PartC
5 {
6     public static void Main( string[] args )
7     {
8         for ( int i = 19; i >= 1; i += 2 )
9             Console.WriteLine( i );
10    } // end Main
11 } // end class PartC

```

19
21
23
25
27
29
...

```

1 // Exercise 6.9c Solution: PartCCorrected.cs
2 using System;
3
4 public class PartCCorrected
5 {
6     public static void Main( string[] args )
7     {
8         for ( int i = 19; i >= 1; i -= 2 )
9             Console.WriteLine( i );
10    } // end Main
11 } // end class PartCCorrected

```

```

19
17
15
13
11
9
7
5
3
1

```

d) The following code should output the even integers from 2 to 100:

```

counter = 2;

do
{
    Console.WriteLine( counter );
    counter += 2;
} While ( counter < 100 );

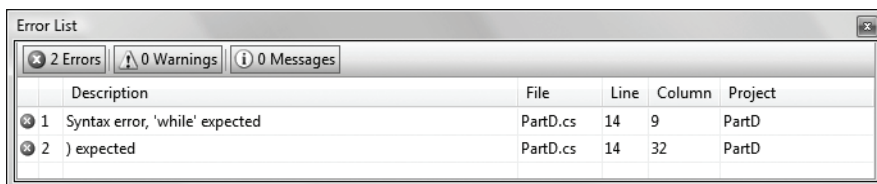
```

ANS: The **W** in **while** should be lowercase. The operator **<** should be **<=**.

```

1 // Exercise 6.9d: PartD.cs
2 using System;
3
4 public class PartD
5 {
6     public static void Main( string[] args )
7     {
8         int counter = 2;
9
10        do
11        {
12            Console.WriteLine( counter );
13            counter += 2;
14        } While ( counter < 100 );
15    } // end Main
16 } // end class PartD

```



```
1 // Exercise 6.9d Solution: PartDCorrected.cs
2 using System;
3
4 public class PartDCorrected
5 {
6     public static void Main( string[] args )
7     {
8         int counter = 2;
9
10        do
11        {
12            Console.WriteLine( counter );
13            counter += 2;
14        } while ( counter <= 100 );
15    } // end Main
16 } // end class PartDCorrected
```

```
2
4
6
...
96
98
100
```

6.10 What does the following application do?

```
1 // Exercise 6.10 Solution: Printing.cs
2 using System;
3
4 public class Printing
5 {
6     public static void Main( string[] args )
7     {
8         for ( int i = 1; i <= 10; i++ )
9         {
10            for ( int j = 1; j <= 5; j++ )
11                Console.Write( '@' );
12
13            Console.WriteLine();
14        } // end outer for
15    } // end Main
16 } // end class Printing
```

ANS:

```

@@@@@
@@@@@
@@@@@
@@@@@
@@@@@
@@@@@
@@@@@
@@@@@
@@@@@
@@@@@

```

6.11 Write an application that finds the smallest of several integers. Assume that the first value read specifies the number of values to input from the user.

ANS:

```

1  // Exercise 6.11 Solution: Small.cs
2  // Application finds the smallest of several integers.
3  using System;
4
5  public class Small
6  {
7      // finds the smallest integer
8      public static void Main( string[] args )
9      {
10         int smallest = 0; // smallest number
11         int number = 0; // number entered by user
12         int integers; // number of integers
13
14         Console.Write( "Enter number of integers: " );
15         integers = Convert.ToInt32( Console.ReadLine() );
16
17         for ( int counter = 1; counter <= integers; counter++ )
18         {
19             Console.Write( "Enter integer: " );
20             number = Convert.ToInt32( Console.ReadLine() );
21
22             if ( counter == 1 )
23                 smallest = number;
24             else if ( number < smallest )
25                 smallest = number;
26         } // end for
27
28         Console.WriteLine( "Smallest integer is: {0}", smallest );
29     } // end Main
30 } // end class Small

```

```

Enter number of integers: 5
Enter integer: 10
Enter integer: -4
Enter integer: 2
Enter integer: 0
Enter integer: 9
Smallest integer is: -4

```

- 6.12** Write an application that calculates the product of the odd integers from 1 to 7.
ANS:

```

1 // Exercise 6.12 Solution: Odd.cs
2 // Application outputs the product of the odd integers from 1 to 7
3 using System;
4
5 public class Odd
6 {
7     public static void Main( string[] args )
8     {
9         int product = 1; // the product of all the odd numbers
10
11         // loop through all odd numbers from 3 to 7
12         for ( int x = 3; x <= 7; x += 2 )
13             product *= x;
14
15         Console.WriteLine( "Product is {0}", product ); // show results
16     } // end Main
17 } // end class Odd

```

```

Product is 105

```

- 6.13** *Factorials* are used frequently in probability problems. The factorial of a positive integer n (written $n!$ and pronounced “ n factorial”) is equal to the product of the positive integers from 1 to n . Write an application that evaluates the factorials of the integers from 1 to 5. Display the results in tabular format. What difficulty might prevent you from calculating the factorial of 20?

ANS: Calculating the factorial of 20 might be difficult because the value of $20!$ exceeds the maximum value that can be stored in an int.

```

1 // Exercise 6.13 Solution: Factorial.cs
2 // Application calculates factorials.
3 using System;
4
5 public class Factorial
6 {
7     public static void Main( string[] args )
8     {
9         Console.WriteLine( "n\t{n!}\n" );
10
11         for ( int number = 1; number <= 5; number++ )
12         {
13             int factorial = 1;

```

```

14
15         for ( int i = 1; i <= number; i++ )
16             factorial *= i;
17
18         Console.WriteLine( "{0}\t{1}", number, factorial );
19     } // end for
20 } // end Main
21 } // end class Factorial

```

n	n!
1	1
2	2
3	6
4	24
5	120

6.14 Modify the compound-interest application of Fig. 6.6 to repeat its steps for interest rates of 5, 6, 7, 8, 9 and 10%. Use a for loop to vary the interest rate.

ANS:

```

1  // Exercise 6.14 Solution: Interest.cs
2  // Calculating compound interest
3  using System;
4
5  public class Interest
6  {
7      public static void Main( string[] args )
8      {
9          decimal amount; // amount on deposit at end of each year
10         decimal principal = 1000M; // initial deposit
11
12         // display statistics for each rate
13         for ( int interestRate = 5; interestRate <= 10; interestRate++ )
14         {
15             double rate = interestRate / 100.0;
16             Console.WriteLine( "\nInterest Rate: {0:F}%", interestRate );
17             Console.WriteLine( "Year\tAmount on deposit" );
18
19             // for each rate, display a ten-year forecast
20             for ( int year = 1; year <= 10; year++ )
21             {
22                 // calculate new amount for specified year
23                 amount = principal *
24                     ( ( decimal ) Math.Pow( 1.0 + rate, year ) );
25
26                 Console.WriteLine( "{0}\t{1:C}", year, amount );
27             } // end year for loop
28         } // end interest for loop
29     } // end Main
30 } // end class Interest

```

Interest Rate: 5.00%	
Year	Amount on deposit
1	\$1,050.00
2	\$1,102.50
3	\$1,157.63
4	\$1,215.51
5	\$1,276.28
6	\$1,340.10
7	\$1,407.10
8	\$1,477.46
9	\$1,551.33
10	\$1,628.89

Interest Rate: 6.00%	
Year	Amount on deposit
1	\$1,060.00
2	\$1,123.60
3	\$1,191.02
4	\$1,262.48
5	\$1,338.23
6	\$1,418.52
7	\$1,503.63
8	\$1,593.85
9	\$1,689.48
10	\$1,790.85

Interest Rate: 7.00%	
Year	Amount on deposit
1	\$1,070.00
2	\$1,144.90
3	\$1,225.04
4	\$1,310.80
5	\$1,402.55
6	\$1,500.73
7	\$1,605.78
8	\$1,718.19
9	\$1,838.46
10	\$1,967.15

Interest Rate: 8.00%	
Year	Amount on deposit
1	\$1,080.00
2	\$1,166.40
3	\$1,259.71
4	\$1,360.49
5	\$1,469.33
6	\$1,586.87
7	\$1,713.82
8	\$1,850.93
9	\$1,999.00
10	\$2,158.92

Interest Rate: 9.00%	
Year	Amount on deposit
1	\$1,090.00
2	\$1,188.10
3	\$1,295.03
4	\$1,411.58
5	\$1,538.62
6	\$1,677.10
7	\$1,828.04
8	\$1,992.56
9	\$2,171.89
10	\$2,367.36

Interest Rate:	10.00%
Year	Amount on deposit
1	\$1,100.00
2	\$1,210.00
3	\$1,331.00
4	\$1,464.10
5	\$1,610.51
6	\$1,771.56
7	\$1,948.72
8	\$2,143.59
9	\$2,357.95
10	\$2,593.74

6.15 Write an application that displays the following patterns separately, one below the other. Use for loops to generate the patterns. All asterisks (*) should be displayed by a single statement of the form `Console.Write('* ');` which causes the asterisks to display side by side. A statement of the form `Console.WriteLine();` can be used to move to the next line. A statement of the form `Console.Write(' ');` can be used to display a space for the last two patterns. There should be no other output statements in the application. [*Hint:* The last two patterns require that each line begin with an appropriate number of blank spaces.]

[illegible]

ANS:

```
1 // Exercise 6.15 Solution: Triangles.cs
2 // Application displays four triangles, one below the other
3 using System;
4
5 public class Triangles
6 {
7     // draws four triangles
8     public static void Main( string[] args )
9     {
```

```
10     int row; // row position
11     int column; // column position
12     int space; // number of spaces to display
13
14     // first triangle
15     for ( row = 1; row <= 10; row++ )
16     {
17         for ( column = 1; column <= row; column++ )
18             Console.Write( "*" );
19
20         Console.WriteLine();
21     } // end outer for
22
23     Console.WriteLine();
24
25     // second triangle
26     for ( row = 10; row >= 1; row-- )
27     {
28         for ( column = 1; column <= row; column++ )
29             Console.Write( "*" );
30
31         Console.WriteLine();
32     } // end outer for
33
34     Console.WriteLine();
35
36     // third triangle
37     for ( row = 10; row >= 1; row-- )
38     {
39         for ( space = 10; space > row; space-- )
40             Console.Write( " " );
41
42         for ( column = 1; column <= row; column++ )
43             Console.Write( "*" );
44
45         Console.WriteLine();
46     } // end outer for
47
48     Console.WriteLine();
49
50     // fourth triangle
51     for ( row = 10; row >= 1; row-- )
52     {
53         for ( space = 1; space < row; space++ )
54             Console.Write( " " );
55
56         for ( column = 10; column >= row; column-- )
57             Console.Write( "*" );
58
59         Console.WriteLine();
60     } // end outer for
61 } // end Main
62 } // end class Triangles
```

```

5 public class BarChart
6 {
7     // draws 3 bars of a bar chart
8     public static void Main( string[] args )
9     {
10         int inputNumber; // number entered by user
11         int counter = 1; // counter for current number
12
13         while ( counter <= 3 )
14         {
15             Console.Write( "\nEnter number between 1 and 30: " );
16             inputNumber = Convert.ToInt32( Console.ReadLine() );
17
18             // displays a bar of asterisks if input is between 1-30
19             if ( inputNumber >= 1 && inputNumber <= 30 )
20             {
21                 for ( int j = 1; j <= inputNumber; j++ )
22                     Console.Write( "*" );
23
24                 Console.WriteLine(); // output an empty line
25                 ++counter;
26             } // end if
27             else
28                 Console.WriteLine(
29                     "Invalid Input\nNumber should be between 1 and 30" );
30         } // end while
31     } // end Main
32 } // end class BarChart

```

```

Enter number between 1 and 30: 10
*****

```

```

Enter number between 1 and 30: 35
Invalid Input
Number should be between 1 and 30

```

```

Enter number between 1 and 30: 20
*****

```

```

Enter number between 1 and 30: 7
*****

```

6.17 A website sells three products whose retail prices are as follows: Product 1, \$2.98; product 2, \$4.50; and product 3, \$9.98. Write an application that reads a series of pairs of numbers as follows:

- a) product number
- b) quantity sold

Your application should use a switch statement to determine the retail price for each product. It should calculate and display the total retail value of all products sold. Use a sentinel-controlled loop to determine when the application should stop looping and display the final results.

ANS:

```

1  // Exercise 6.17 Solution: Sales.cs
2  // Application calculates sales, based on an input of product
3  // number and quantity sold
4  using System;
5
6  public class Sales
7  {
8      // calculates sales for 3 products
9      public static void Main( string[] args )
10     {
11         decimal product1 = 0M; // amount sold of first product
12         decimal product2 = 0M; // amount sold of second product
13         decimal product3 = 0M; // amount sold of third product
14
15         int productId = 1; // current product id number
16
17         // ask user for product number until flag value entered
18         while ( productId != 0 )
19         {
20             // determine the product chosen
21             Console.Write(
22                 "Enter product number (1-3) (0 to stop): " );
23             productId = Convert.ToInt32( Console.ReadLine() );
24
25             if ( productId >= 1 && productId <= 3 )
26             {
27                 // determine the number sold of the item
28                 Console.Write( "Enter quantity sold: " );
29                 int quantity = Convert.ToInt32( Console.ReadLine() );
30
31                 // increment the total for the item by the
32                 // price times the quantity sold
33                 switch ( productId )
34                 {
35                     case 1:
36                         product1 += quantity * 2.98M;
37                         break;
38                     case 2:
39                         product2 += quantity * 4.50M;
40                         break;
41                     case 3:
42                         product3 += quantity * 9.98M;
43                         break;
44                 } // end switch
45             } // end if
46             else if ( productId != 0 )
47                 Console.WriteLine(
48                     "Product number must be between 1 and 3 or 0 to stop" );
49         } // end while
50
51         // display summary
52         Console.WriteLine( "\nProduct 1: {0:C}", product1 );

```

```

53     Console.WriteLine( "Product 2: {0:C}", product2 );
54     Console.WriteLine( "Product 3: {0:C}", product3 );
55 } // end Main
56 } // end class Sales

```

```

Enter product number (1-3) (0 to stop): 1
Enter quantity sold: 100
Enter product number (1-3) (0 to stop): 2
Enter quantity sold: 200
Enter product number (1-3) (0 to stop): 1
Enter quantity sold: 50
Enter product number (1-3) (0 to stop): 3
Enter quantity sold: 10
Enter product number (1-3) (0 to stop): 1
Enter quantity sold: 100
Enter product number (1-3) (0 to stop): 0

```

```

Product 1: $745.00
Product 2: $900.00
Product 3: $99.80

```

6.18 In the future, you may work with other programming languages that do not have a type like `decimal` which supports precise monetary calculations. In those languages, you should perform such calculations using integers. Modify the application in Fig. 6.6 to use only integers to calculate the compound interest. Treat all monetary amounts as integral numbers of pennies. Then break the result into its dollars and cents portions by using the division and remainder operations, respectively. Insert a period between the dollars and the cents portions when you display the results.

ANS:

```

1  // Exercise 6.18: Interest.cs
2  // Calculating compound interest.
3  using System;
4
5  public class Interest
6  {
7      public static void Main( string[] args )
8      {
9          int amount; // amount on deposit at end of each year
10         int principal = 100000; // initial amount (number of pennies)
11         double rate = 0.05; // interest rate
12
13         // display the headers
14         Console.WriteLine( "{0}{1,20}", "Year", "Amount on deposit" );
15
16         // calculate amount on deposit for each of ten years
17         for ( int year = 1; year <= 10; year++ )
18         {
19             // calculate new amount for specified year
20             amount = ( int ) ( principal * Math.Pow( 1.0 + rate, year ) );
21
22             int dollars = amount / 100; // calculate dollars portion
23             int cents = amount % 100; // calculate cents portion

```

```

24
25         // output the year, dollars and cents
26         Console.WriteLine( "{0,2}{1,19}.{2:D2}", year, dollars, cents );
27     } // end for
28 } // end Main
29 } // end class Interest

```

Year	Amount on deposit
1	1050.00
2	1102.50
3	1157.62
4	1215.50
5	1276.28
6	1340.09
7	1407.10
8	1477.45
9	1551.32
10	1628.89

6.19 Assume that $i = 1$, $j = 2$, $k = 3$ and $m = 2$. What does each of the following statements display?

- a) `Console.WriteLine(i == 1);`
ANS: true.
- b) `Console.WriteLine(j == 3);`
ANS: false.
- c) `Console.WriteLine((i >= 1) && (j < 4));`
ANS: true.
- d) `Console.WriteLine((m <= 99) & (k < m));`
ANS: false.
- e) `Console.WriteLine((j >= i) || (k == m));`
ANS: true.
- f) `Console.WriteLine((k + m < j) | (3 - j >= k));`
ANS: false.
- g) `Console.WriteLine(!(k > m));`
ANS: false.

```

1 // Exercise 6.19 Solution: Mystery.cs
2 // Displaying conditional expressions outputs 'true' or 'false'.
3 using System;
4
5 public class Mystery
6 {
7     public static void Main( string[] args )
8     {
9         int i = 1;
10        int j = 2;
11        int k = 3;
12        int m = 2;
13
14        // part a
15        Console.Write( "Part a: " );
16        Console.WriteLine( i == 1 );

```

```

17
18     // part b
19     Console.Write( "Part b: " );
20     Console.WriteLine( j == 3 );
21
22     // part c
23     Console.Write( "Part c: " );
24     Console.WriteLine( ( i >= 1 ) && ( j < 4 ) );
25
26     // part d
27     Console.Write( "Part d: " );
28     Console.WriteLine( ( m <= 99 ) & ( k < m ) );
29
30     // part e
31     Console.Write( "Part e: " );
32     Console.WriteLine( ( j >= i ) || ( k == m ) );
33
34     // part f
35     Console.Write( "Part f: " );
36     Console.WriteLine( ( k + m < j ) | ( 3 - j >= k ) );
37
38     // part g
39     Console.Write( "Part g: " );
40     Console.WriteLine( !( k > m ) );
41 } // end Main
42 } // end class Mystery

```

```

Part a: True
Part b: False
Part c: True
Part d: False
Part e: True
Part f: False
Part g: False

```

6.20 Calculate the value of π from the infinite series

$$\pi = 4 - \frac{4}{3} + \frac{4}{5} - \frac{4}{7} + \frac{4}{9} - \frac{4}{11} + \dots$$

Display a table that shows the value of π approximated by computing one term of this series, by two terms, by three terms, and so on. How many terms of this series do you have to use before you first get 3.14? 3.141? 3.1415? 3.14159?

ANS: [Note: the calculation starts to converge around 3.14 at 119 terms, but does not really approach 3.141 until 1688 terms.]

```

1 // Exercise 6.20 Solution: Pi.cs
2 // Application calculates Pi from the infinite series.
3 using System;
4
5 public class Pi
6 {

```

```

7   public static void Main( string[] args )
8   {
9       double piValue = 0.0; // current approximation of pi
10      double numerator = 4.0; // numerator of fraction
11      double denominator = 1.0; // denominator
12      int accuracy; // number of terms in series
13
14      // prompt user and input number of terms
15      Console.Write( "Enter the number of terms: " );
16      accuracy = Convert.ToInt32( Console.ReadLine() );
17
18      // display table
19      Console.WriteLine( "Term\t\tPi" );
20
21      for ( int term = 1; term <= accuracy; term++ )
22      {
23          if ( term % 2 != 0 )
24              piValue += numerator / denominator;
25          else
26              piValue -= numerator / denominator;
27
28          Console.WriteLine( "{0}\t\t{1:F16}", term, piValue );
29          denominator += 2.0;
30      } // end for
31  } // end Main
32 } // end class Pi

```

```

Enter the number of terms: 119
Term      Pi
1          4.0000000000000000
2          2.6666666666666700
3          3.4666666666666700
...
117        3.1501395060584100
118        3.1331182294626700
119        3.1499958665934700

```

```

Enter the number of terms: 50000
Term      Pi
1          4.0000000000000000
2          2.6666666666666700
3          3.4666666666666700
...
49998     3.1415726527897500
49999     3.1416126539897900
50000     3.1415726535897800

```

6.21 (*Pythagorean Triples*) A right triangle can have sides whose lengths are all integers. The set of three integer values for the lengths of the sides of a right triangle is called a Pythagorean triple. The lengths of the three sides must satisfy the relationship that the sum of the squares of two of the sides is equal to the square of the hypotenuse. Write an application to find all Pythagorean triples for side1, side2 and the hypotenuse, all no larger than 500. Use a triple-nested for loop that tries

all possibilities. This method is an example of “brute-force” computing. You’ll learn in more advanced computer science courses that there are many interesting problems for which there is no known algorithmic approach other than using sheer brute force.

ANS:

```

1 // Exercise 6.21 Solution: Triples.cs
2 // Application calculates Pythagorean triples
3 using System;
4
5 public class Triples
6 {
7     public static void Main( string[] args )
8     {
9         // declare the three sides of a triangle
10        int side1;
11        int side2;
12        int hypotenuse;
13
14        // display a table header
15        Console.WriteLine( "{0,8}{1,8}{2,12}",
16            "side1", "side2", "hypotenuse" );
17
18        for ( side1 = 1; side1 <= 500; side1++ )
19        {
20            for ( side2 = 1; side2 <= 500; side2++ )
21            {
22                for ( hypotenuse = 1; hypotenuse <= 500; hypotenuse++ )
23                {
24                    // use Pythagorean Theorem to display right triangles
25                    if ( ( side1 * side1 ) + ( side2 * side2 ) ==
26                        ( hypotenuse * hypotenuse ) )
27                    {
28                        Console.WriteLine( "{0,8}{1,8}{2,12}",
29                            side1, side2, hypotenuse );
30                    } // end if
31                } // end for (hypotenuse)
32            } // end for (side2)
33        } // end for (side1)
34    } // end Main
35 } // end class Triples

```

side1	side2	hypotenuse
3	4	5
4	3	5
5	12	13
6	8	10
...		
480	88	488
480	108	492
480	140	500
483	44	485

6.22 Modify Exercise 6.15 to combine your code from the four separate triangles of asterisks such that all four patterns display side by side. Make clever use of nested for loops.

ANS:

```

1 // Exercise 6.22 Solution: Triangles.cs
2 // Application draws four triangles side by side
3 using System;
4
5 public class Triangles
6 {
7     // draws four triangles
8     public static void Main( string[] args )
9     {
10         int row; // current row
11         int column; // current column
12         int space; // number of spaces outputted
13
14         // draw one row at a time, tabbing between triangles
15         for ( row = 1; row <= 10; row++ )
16         {
17             // triangle one
18             for ( column = 1; column <= row; column++ )
19                 Console.Write( "*" );
20
21             for ( space = 1; space <= 10 - row; space++ )
22                 Console.Write( " " );
23
24             Console.Write( "\t" );
25
26             // triangle two
27             for ( column = 10; column >= row; column-- )
28                 Console.Write( "*" );
29
30             for ( space = 1; space < row; space++ )
31                 Console.Write( " " );
32
33             Console.Write( "\t" );
34
35             // triangle three
36             for ( space = 1; space < row; space++ )
37                 Console.Write( " " );
38
39             for ( column = 10; column >= row; column-- )
40                 Console.Write( "*" );
41
42             Console.Write( "\t" );
43
44             // triangle four
45             for ( space = 10; space > row; space-- )
46                 Console.Write( " " );
47
48             for ( column = 1; column <= row; column++ )
49                 Console.Write( "*" );
50

```

```

19
20     for ( stars = 1; stars <= ( 2 * row ) - 1; stars++ )
21         Console.Write( "*" );
22
23     Console.WriteLine();
24 } // end outer for
25
26 // bottom half (last four rows)
27 for ( row = 4; row >= 1; row-- )
28 {
29     for ( spaces = 5; spaces > row; spaces-- )
30         Console.Write( " " );
31
32     for ( stars = 1; stars <= ( 2 * row ) - 1; stars++ )
33         Console.Write( "*" );
34
35     Console.WriteLine();
36 } // end outer for
37 } // end Main
38 } // end class Diamond

```

```

      *
     ***
    *****
   *******
  *********
 *****
  *****
   *****
    ***
     *

```

6.24 Modify the application you wrote in Exercise 6.23 to read an odd number in the range 1 to 19 to specify the number of rows in the diamond. Your application should then display a diamond of the appropriate size.

ANS:

```

1  // Exercise 6.24 Solution: Diamond.cs
2  // Application displays a diamond of a user-specified size
3  using System;
4
5  public class Diamond
6  {
7      // draws a diamond of asterisks
8      public static void Main( string[] args )
9      {
10         int row; // current row
11         int stars; // number of stars
12         int spaces; // number of spaces
13         int numRows; // number of rows
14
15         // prompt for number of rows until within range
16         Console.Write( "Enter number of rows (odd number 1 to 19): " );

```

```

17 numRows = Convert.ToInt32( Console.ReadLine() );
18
19 while ( ( numRows > 19 ) || ( numRows < 0 ) ||
20      ( numRows % 2 == 0 ) )
21 {
22     Console.WriteLine( "Invalid input." );
23     Console.Write( "Enter number of rows (odd number 1 to 19): " );
24     numRows = Convert.ToInt32( Console.ReadLine() );
25 } // end while
26
27 // top half
28 for ( row = 1; row < ( numRows / 2 ) + 1; row++ )
29 {
30     for ( spaces = numRows / 2; spaces >= row; spaces-- )
31         Console.Write( " " );
32
33     for ( stars = 1; stars <= ( 2 * row ) - 1; stars++ )
34         Console.Write( "*" );
35
36     Console.WriteLine();
37 } // end outer for
38
39 // bottom half, note that the first clause of the for
40 // loop isn't needed; row is already defined
41 for ( ; row >= 1; row-- )
42 {
43     for ( spaces = numRows / 2; spaces >= row; spaces-- )
44         Console.Write( " " );
45
46     for ( stars = 1; stars <= ( 2 * row ) - 1; stars++ )
47         Console.Write( "*" );
48
49     Console.WriteLine();
50 } // end outer for
51 } // end Main
52 } // end class Diamond

```

[illegible]

6.25 A criticism of the `break` statement and the `continue` statement is that each is unstructured. Actually, `break` and `continue` statements can always be replaced by structured statements, although doing so can be awkward. Describe in general how you would remove any `break` statement from a loop in an application and replace it with a structured equivalent. [*Hint:* The `break` statement exits a loop from the body of the loop. The other way to exit is by failing the loop-continuation test. Consider using in the loop-continuation test a second test that indicates “early exit because of a ‘break’ condition.”] Use the technique you develop here to remove the `break` statement from the application in Fig. 6.12.

ANS: A loop can be written without a `break` by placing in the loop-continuation test a second test that indicates “early exit because of a ‘break’ condition.” Alternatively, the `break` can be replaced by a statement that makes the original loop-continuation test immediately false, so that the loop terminates.

```

1 // Exercise 6.25: WithoutBreak.cs
2 // Terminating a loop without break.
3 using System;
4
5 public class WithoutBreak
6 {
7     public static void Main( string[] args )
8     {
9         int count; // control variable
10        bool breakOut = false; // control variable for second test
11
12        // loop 10 times
13        for ( count = 1; ( count <= 10 ) && ( !breakOut ); count++ )
14        {
15            Console.Write( "{0} ", count );
16
17            if ( count == 4 ) // if count is 4,
18                breakOut = true; // set breakOut to terminate loop
19        } // end for
20
21        Console.WriteLine( "\n{0}\n{1}", "Broke out of loop",
22                           "because loop-continuation test ( !breakOut ) failed." );
23    } // end Main
24 } // end class WithoutBreak

```

```

1 2 3 4
Broke out of loop
because loop-continuation test ( !breakOut ) failed.

```

6.26 What does the following code segment do?

```

for ( int i = 1; i <= 5; i++ )
{
    for ( int j = 1; j <= 3; j++ )
    {
        for ( int k = 1; k <= 4; k++ )
            Console.Write( '*' );

        Console.WriteLine();
    } // end middle for

    Console.WriteLine();
} // end outer for

```

ANS:

```

1 // Exercise 6.26 Solution: Mystery.cs
2 // Displays 5 groups of 3 lines, each containing 4 asterisks.
3 using System;
4
5 public class Mystery
6 {
7     public static void Main( string[] args )
8     {
9         for ( int i = 1; i <= 5; i++ )
10        {
11            for ( int j = 1; j <= 3; j++ )
12            {
13                for ( int k = 1; k <= 4; k++ )
14                    Console.Write( '*' );
15
16                Console.WriteLine();
17            } // end middle for
18
19            Console.WriteLine();
20        } // end outermost for
21    } // end Main
22 } // end class Mystery

```

```

*****
*****
*****

*****
*****
*****

*****
*****
*****

*****
*****
*****

*****
*****
*****

```

6.27 Describe in general how you would remove any `continue` statement from a loop in an application and replace it with some structured equivalent. Use the technique you develop here to remove the `continue` statement from the application in Fig. 6.13.

ANS: A loop can be rewritten without a `continue` statement by moving all the code that appears in the body of the loop after the `continue` statement into an `if` statement that tests for the opposite of the `continue` condition. Thus, the code that was originally after the `continue` statement executes only when the `if` statement's conditional expression is true (i.e., the "continue" condition is false). When the "continue" condi-

tion is true, the body of the `if` does not execute and the application “continues” to the next iteration of the loop by not executing the remaining code in the loop’s body. The answer shown here removes the `continue` statement from the `if` statement.

```

1 // Exercise 6.27 Solution: ContinueTest.cs
2 // Alternative to the continue statement in a for statement
3 using System;
4
5 public class ContinueTest
6 {
7     public static void Main( string[] args )
8     {
9         for ( int count = 1; count <= 10; count++ ) // loop 10 times
10             if ( count != 5 ) // if count is not 5
11                 Console.Write( "{0} ", count ); // display
12
13         Console.WriteLine( "\nUsed if statement to skip displaying 5" );
14     } // end Main
15 } // end class ContinueTest

```

```

1 2 3 4 6 7 8 9 10
Used if statement to skip displaying 5

```

Making a Difference

6.28 (Global Warming Facts Quiz) The controversial issue of global warming has been widely publicized by the film *An Inconvenient Truth*, featuring former Vice President Al Gore. Mr. Gore and a U.N. network of scientists, the Intergovernmental Panel on Climate Change, shared the 2007 Nobel Peace Prize in recognition of “their efforts to build up and disseminate greater knowledge about man-made climate change.” Research *both* sides of the global warming issue online (you might want to search for phrases like “global warming skeptics”). Create a five-question multiple-choice quiz on global warming, each question having four possible answers (numbered 1–4). Be objective and try to fairly represent both sides of the issue. Next, write an application that administers the quiz, calculates the number of correct answers (zero through five) and returns a message to the user. If the user correctly answers five questions, print “Excellent”; if four, print “Very good”; if three or fewer, print “Time to brush up on your knowledge of global warming,” and include a list of some of the websites where you found your facts.

ANS:

```

1 // Exercise 6.28 Solution: GlobalWarmingQuiz.cs
2 // Administers global warming quiz.
3 using System;
4
5 class GlobalWarmingQuiz
6 {
7     public static void Main( string[] args )
8     {
9         int answer; // answer user gave for current question
10        int score = 0; // number of correct answers
11

```

```
12 // first question
13 Console.WriteLine(
14     "By how much have average temperatures risen since 1880?");
15 Console.WriteLine( "1: 0.4 degrees F" );
16 Console.WriteLine( "2: 1.4 degrees F" );
17 Console.WriteLine( "3: 2.4 degrees F" );
18 Console.WriteLine( "4: 3.4 degrees F" );
19
20 Console.Write( "> " ); // display prompt
21 answer = Convert.ToInt32( Console.ReadLine() ); // read answer
22
23 if ( answer == 2 )
24 {
25     ++score; // increment score if answer is correct
26     Console.WriteLine( "Correct!\n" );
27 } // end if
28 else
29     Console.WriteLine( "Incorrect. The correct answer was 2\n" );
30
31 // second question
32 Console.WriteLine(
33     "Montana's Glacier National Park had 150 glaciers in 1910." );
34 Console.WriteLine( "How many does it have now?" );
35 Console.WriteLine( "1: 0" );
36 Console.WriteLine( "2: 7" );
37 Console.WriteLine( "3: 17" );
38 Console.WriteLine( "4: 27" );
39
40 Console.Write( "> " ); // display prompt
41 answer = Convert.ToInt32( Console.ReadLine() ); // read answer
42
43 if ( answer == 4 )
44 {
45     ++score; // increment score if answer is correct
46     Console.WriteLine( "Correct!\n" );
47 } // end if
48 else
49     Console.WriteLine( "Incorrect. The correct answer was 4\n" );
50
51 // third question
52 Console.WriteLine( "What is the most abundant greenhouse gas?" );
53 Console.WriteLine( "1: water vapor" );
54 Console.WriteLine( "2: carbon dioxide" );
55 Console.WriteLine( "3: methane" );
56 Console.WriteLine( "4: carbon monoxide" );
57
58 Console.Write( "> " ); // display prompt
59 answer = Convert.ToInt32( Console.ReadLine() ); // read answer
60
61 if ( answer == 1 )
62 {
63     ++score; // increment score if answer is correct
64     Console.WriteLine( "Correct!\n" );
65 } // end if
```

```
66     else
67         Console.WriteLine( "Incorrect. The correct answer was 1\n" );
68
69     // fourth question
70     Console.WriteLine(
71         "Which of these should you NOT do to help stop global warming?" );
72     Console.WriteLine( "1: Use less hot water" );
73     Console.WriteLine( "2: Reuse your shopping bag" );
74     Console.WriteLine( "3: Plant a tree" );
75     Console.WriteLine( "4: Take a bath instead of a shower" );
76
77     Console.Write( "> " ); // display prompt
78     answer = Convert.ToInt32( Console.ReadLine() ); // read answer
79
80     if ( answer == 4 )
81     {
82         ++score; // increment score if answer is correct
83         Console.WriteLine( "Correct!\n" );
84     } // end if
85     else
86         Console.WriteLine( "Incorrect. The correct answer was 4\n" );
87
88     // fifth question
89     Console.WriteLine(
90         "Which of these should you NOT do to help stop global warming?" );
91     Console.WriteLine( "1: Buy more frozen foods" );
92     Console.WriteLine( "2: Fly less" );
93     Console.WriteLine( "3: Use a clothesline instead of a dryer" );
94     Console.WriteLine( "4: Cover pots while cooking" );
95
96     Console.Write( "> " ); // display prompt
97     answer = Convert.ToInt32( Console.ReadLine() ); // read answer
98
99     if ( answer == 1 )
100     {
101         ++score; // increment score if answer is correct
102         Console.WriteLine( "Correct!\n" );
103     } // end if
104     else
105         Console.WriteLine( "Incorrect. The correct answer was 1\n" );
106
107     // display message based on number of correct answers
108     if ( score == 5 )
109         Console.WriteLine( "Excellent" );
110     else if ( score == 4 )
111         Console.WriteLine( "Very good" );
112     else
113     {
114         Console.WriteLine(
115             "Time to brush up on your knowledge of global warming:" );
116         Console.WriteLine(
117             "http://news.nationalgeographic.com/news/2004/12/" +
118             "1206_041206_global_warming.html" );

```

```

119         Console.WriteLine(
120             "http://lwf.ncdc.noaa.gov/oa/climate/gases.html" );
121         Console.WriteLine(
122             "http://globalwarming-facts.info/50-tips.html" );
123     } // end else
124 } // end Main
125 } // end class GlobalWarmingQuiz

```

ANS:

By how much have average temperatures risen since 1880?

- 1: 0.4 degrees F
- 2: 1.4 degrees F
- 3: 2.4 degrees F
- 4: 3.4 degrees F

> 3

Incorrect. The correct answer was 2

Montana's Glacier National Park had 150 glaciers in 1910.

How many does it have now?

- 1: 0
- 2: 7
- 3: 17
- 4: 27

> 4

Correct!

What is the most abundant greenhouse gas?

- 1: water vapor
- 2: carbon dioxide
- 3: methane
- 4: carbon monoxide

> 2

Incorrect. The correct answer was 1

Which of these should you NOT do to help stop global warming?

- 1: Use less hot water
- 2: Reuse your shopping bag
- 3: Plant a tree
- 4: Take a bath instead of a shower

> 4

Correct!

Which of these should you NOT do to help stop global warming?

- 1: Buy more frozen foods
- 2: Fly less
- 3: Use a clothesline instead of a dryer
- 4: Cover pots while cooking

> 1

Correct!

Time to brush up on your knowledge of global warming:

[http://news.nationalgeographic.com/news/2004/12/](http://news.nationalgeographic.com/news/2004/12/1206_041206_global_warming.html)

[1206_041206_global_warming.html](http://news.nationalgeographic.com/news/2004/12/1206_041206_global_warming.html)

<http://lwf.ncdc.noaa.gov/oa/climate/gases.html>

<http://globalwarming-facts.info/50-tips.html>

6.29 (*Tax Plan Alternatives; The “FairTax”*) There are many proposals to make taxation fairer. Check out the FairTax initiative in the United States at

www.fairtax.org/site/PageServer?pagename=calculator

Research how the proposed FairTax works. One suggestion is to eliminate income taxes and most other taxes in favor of a 23% consumption tax on all products and services that you buy. Some FairTax opponents question the 23% figure and say that because of the way the tax is calculated, it would be more accurate to say the rate is 30%—check this carefully. Write a program that prompts the user to enter expenses in various expense categories they have (e.g., housing, food, clothing, transportation, education, health care, vacations), then prints the estimated FairTax that person would pay.

ANS:

```

1  // Exercise 6.29 Solution: FairTax.cs
2  // Calculates the tax the user would owe under the FairTax plan.
3  using System;
4
5  class Program
6  {
7      public static void Main( string[] args )
8      {
9          int amount; // amount the user pays for each expense
10         int total = 0;
11
12         // display the expense categories for the user
13         Console.WriteLine( "Welcome to the Fair Tax Calculator!" );
14         Console.WriteLine( "Here are some common expense categories:" );
15         Console.WriteLine(
16             "1: Housing, 2: Food, 3: Clothing, 4: Transportation," );
17         Console.WriteLine( "5: Education, 6: Health care, 7: Vacations\n" );
18
19         // prompt for and input the amount the user paid in each category
20         for ( int i = 1; i <= 7; ++i )
21         {
22             Console.WriteLine( "Total spending for category {0}: ", i );
23             amount = Convert.ToInt32( Console.ReadLine() );
24             total += amount;
25         } // end for
26
27         // calculate and display tax
28         decimal tax = total * 0.3M;
29         Console.WriteLine( "Your total Fair Tax would be {0:C}", tax );
30     } // end Main
31 } // end class FairTax

```

Welcome to the Fair Tax Calculator!
Here are some common expense categories:
1: Housing, 2: Food, 3: Clothing, 4: Transportation,
5: Education, 6: Health care, 7: Vacations

Total spending for category 1:
36000
Total spending for category 2:
12000
Total spending for category 3:
6000
Total spending for category 4:
5200
Total spending for category 5:
10000
Total spending for category 6:
12600
Total spending for category 7:
2000
Your total Fair Tax would be \$25,140.00

Methods: A Deeper Look Solutions

7

E pluribus unum.
(*One composed of many.*)
—Virgil

*O! call back yesterday, bid time
return.*
—William Shakespeare

Call me Ishmael.
—Herman Melville

Answer me in one word.
—William Shakespeare

*There is a point at which
methods devour themselves.*
—Frantz Fanon

Objectives

In this chapter you'll learn:

- How static methods and variables are associated with classes rather than objects.
- How the method call/return mechanism is supported by the method-call stack.
- How to use random-number generation to implement game-playing applications.
- How the visibility of declarations is limited to specific parts of applications.
- How to create overloaded methods.
- How to use optional and named parameters.
- What recursive methods are.
- Passing method arguments by value and by reference.

Self-Review Exercises

7.1 Fill in the blanks in each of the following statements:

a) A method is invoked with a(n) _____.

ANS: method call.

b) A variable known only within the method in which it is declared is called a(n) _____.

ANS: local variable.

c) The _____ statement in a called method can be used to pass the value of an expression back to the calling method.

ANS: return

d) The keyword _____ indicates that a method does not return a value.

ANS: void

e) Data can be added to or removed from only the _____ of a stack.

ANS: top.

f) Stacks are known as _____ data structures—the last item pushed (inserted) on the stack is the first item popped off (removed from) the stack.

ANS: last-in-first-out (LIFO).

g) The three ways to return control from a called method to a caller are _____, _____, and _____.

ANS: return; or return expression; or encountering the closing right brace of a method.

h) An object of class _____ produces pseudorandom numbers.

ANS: Random.

i) The program-execution stack contains the memory for local variables on each invocation of a method during an application's execution. This data, stored as a portion of the program-execution stack, is known as the _____ or _____ of the method call.

ANS: activation record, stack frame.

j) If there are more method calls than can be stored on the program-execution stack, an error known as a(n) _____ occurs.

ANS: stack overflow.

k) The _____ of a declaration is the portion of an application that can refer to the entity in the declaration by its unqualified name.

ANS: scope.

l) It is possible to have several methods with the same name that each operate on different types or numbers of arguments. This feature is called method _____.

ANS: overloading.

m) The program execution stack is also referred to as the _____ stack.

ANS: method call.

n) A method that calls itself either directly or indirectly is a(n) _____ method.

ANS: recursive

o) A recursive method typically has two components: one that provides a means for the recursion to terminate by testing for a(n) _____ case and one that expresses the problem as a recursive call for a slightly simpler problem than does the original call.

ANS: base

7.2 For the class Craps in Fig. 7.9, state the scope of each of the following entities:

a) the variable randomNumbers.

ANS: class body.

b) the variable die1.

ANS: block that defines method RollDice's body.

c) the method RollDice.

ANS: class body.

d) the method Main.

ANS: class body.

e) the variable sumOfDice.

ANS: block that defines method Play's body.

7.3 Write an application that tests whether the examples of the Math class method calls shown in Fig. 7.2 actually produce the indicated results.

ANS:

```

1  // Exercise 7.3 Solution: MathTest.cs
2  // Testing the Math class methods.
3  using System;
4
5  public class MathTest
6  {
7      public static void Main( string[] args )
8      {
9          Console.WriteLine( "Math.Abs( 23.7 ) = {0}", Math.Abs( 23.7 ) );
10         Console.WriteLine( "Math.Abs( 0.0 ) = {0}", Math.Abs( 0.0 ) );
11         Console.WriteLine( "Math.Abs( -23.7 ) = {0}", Math.Abs( -23.7 ) );
12         Console.WriteLine( "Math.Ceiling( 9.2 ) = {0}",
13             Math.Ceiling( 9.2 ) );
14         Console.WriteLine( "Math.Ceiling( -9.8 ) = {0}",
15             Math.Ceiling( -9.8 ) );
16         Console.WriteLine( "Math.Cos( 0.0 ) = {0}", Math.Cos( 0.0 ) );
17         Console.WriteLine( "Math.Exp( 1.0 ) = {0}", Math.Exp( 1.0 ) );
18         Console.WriteLine( "Math.Exp( 2.0 ) = {0}", Math.Exp( 2.0 ) );
19         Console.WriteLine( "Math.Floor( 9.2 ) = {0}", Math.Floor( 9.2 ) );
20         Console.WriteLine( "Math.Floor( -9.8 ) = {0}",
21             Math.Floor( -9.8 ) );
22         Console.WriteLine( "Math.Log( Math.E ) = {0}",
23             Math.Log( Math.E ) );
24         Console.WriteLine( "Math.Log( Math.E * Math.E ) = {0}",
25             Math.Log( Math.E * Math.E ) );
26         Console.WriteLine( "Math.Max( 2.3, 12.7 ) = {0}",
27             Math.Max( 2.3, 12.7 ) );
28         Console.WriteLine( "Math.Max( -2.3, -12.7 ) = {0}",
29             Math.Max( -2.3, -12.7 ) );
30         Console.WriteLine( "Math.Min( 2.3, 12.7 ) = {0}",
31             Math.Min( 2.3, 12.7 ) );
32         Console.WriteLine( "Math.Min( -2.3, -12.7 ) = {0}",
33             Math.Min( -2.3, -12.7 ) );
34         Console.WriteLine( "Math.Pow( 2.0, 7.0 ) = {0}",
35             Math.Pow( 2.0, 7.0 ) );
36         Console.WriteLine( "Math.Pow( 9.0, 0.5 ) = {0}",
37             Math.Pow( 9.0, 0.5 ) );
38         Console.WriteLine( "Math.Sin( 0.0 ) = {0}", Math.Sin( 0.0 ) );
39         Console.WriteLine( "Math.Sqrt( 900.0 ) = {0}",
40             Math.Sqrt( 900.0 ) );
41         Console.WriteLine( "Math.Sqrt( 9.0 ) = {0}", Math.Sqrt( 9.0 ) );
42         Console.WriteLine( "Math.Tan( 0.0 ) = {0}", Math.Tan( 0.0 ) );
43     } // end Main
44 } // end class MathTest

```

```

Math.Abs( 23.7 ) = 23.7
Math.Abs( 0.0 ) = 0
Math.Abs( -23.7 ) = 23.7
Math.Ceiling( 9.2 ) = 10
Math.Ceiling( -9.8 ) = -9
Math.Cos( 0.0 ) = 1
Math.Exp( 1.0 ) = 2.71828182845905
Math.Exp( 2.0 ) = 7.38905609893065
Math.Floor( 9.2 ) = 9
Math.Floor( -9.8 ) = -10
Math.Log( Math.E ) = 1
Math.Log( Math.E * Math.E ) = 2
Math.Max( 2.3, 12.7 ) = 12.7
Math.Max( -2.3, -12.7 ) = -2.3
Math.Min( 2.3, 12.7 ) = 2.3
Math.Min( -2.3, -12.7 ) = -12.7
Math.Pow( 2.0, 7.0 ) = 128
Math.Pow( 9.0, 0.5 ) = 3
Math.Sin( 0.0 ) = 0
Math.Sqrt( 900.0 ) = 30
Math.Sqrt( 9.0 ) = 3
Math.Tan( 0.0 ) = 0

```

7.4 Give the method header for each of the following methods:

- a) Method Hypotenuse, which takes two double-precision, floating-point arguments side1 and side2 and returns a double-precision, floating-point result.

ANS: **double** Hypotenuse(**double** side1, **double** side2)

- b) Method Smallest, which takes three integers x, y and z and returns an integer.

ANS: **int** Smallest(**int** x, **int** y, **int** z)

- c) Method Instructions, which does not take any arguments and does not return a value.

[Note: Such methods are commonly used to display instructions to a user.]

ANS: **void** Instructions()

- d) Method IntToDouble, which takes integer argument number and returns a double value.

ANS: **double** IntToDouble(**int** number)

7.5 Find the error in each of the following code segments. Explain how to correct the error.

- a) **void** G()

```

{
    Console.WriteLine( "Inside method G" );
    void H()
    {
        Console.WriteLine( "Inside method H" );
    }
}

```

```

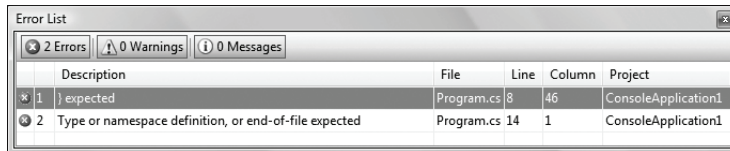
1 // Exercise 7.5a Solution: PartAError.cs
2 using System;
3
4 public class PartAError
5 {
6     void G()
7     {

```

```

8      Console.WriteLine( "Inside method G" );
9      void H()
10     {
11         Console.WriteLine( "Inside method H" );
12     } // end method H
13 } // end method G
14 } // end class PartAError

```



ANS: Error: Method H is declared within method G.
 Correction: Move the declaration of H outside the declaration of G.

```

1  // Exercise 7.5a Solution: PartACorrect.cs
2  using System;
3
4  public class PartACorrect
5  {
6      void G()
7      {
8          Console.WriteLine( "Inside method G" );
9      } // end method G
10
11     void H()
12     {
13         Console.WriteLine( "Inside method H" );
14     } // end method H
15 } // end class PartACorrect

```

```

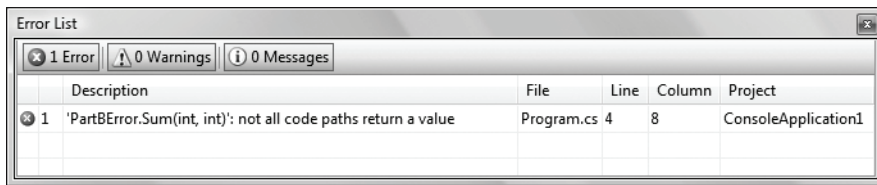
b) int Sum( int x, int y )
    {
        int result;
        result = x + y;
    }

```

```

1  // Exercise 7.5b Solution: PartBError.cs
2  public class PartBError
3  {
4      int Sum( int x, int y )
5      {
6          int result;
7          result = x + y;
8      } // end method Sum
9  } // end class PartBError

```



ANS: Error: The method is supposed to return an integer, but does not.

Correction: Delete variable `result` and place the statement

return `x + y`;

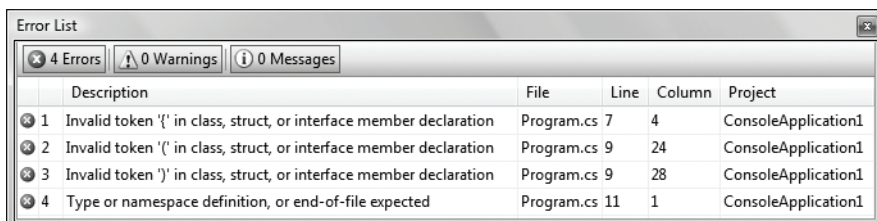
in the method, or add the following statement at the end of the method body:

return `result`;

```
1 // Exercise 7.5b Solution: PartBCorrect.cs
2 public class PartBCorrect
3 {
4     int Sum( int x, int y )
5     {
6         return x + y;
7     } // end method Sum
8 } // end class PartBCorrect
```

```
c) void F( float a );
{
    float a;
    Console.WriteLine( a );
}
```

```
1 // Exercise 7.5c Solution: PartCError.cs
2 using System;
3
4 public class PartCError
5 {
6     void F( float a );
7     {
8         float a;
9         Console.WriteLine( a );
10    } // end method F
11 } // end class PartCError
```



ANS: Error: The semicolon after the right parenthesis of the parameter list is incorrect, and the parameter a should not be redeclared in the method.

Correction: Delete the semicolon after the right parenthesis of the parameter list, and delete the declaration `float a;`.

```

1 // Exercise 7.5c Solution: PartCCorrect.cs
2 using System;
3
4 public class PartCCorrect
5 {
6     void F( float a )
7     {
8         Console.WriteLine( a );
9     } // end method F
10 } // end class PartCCorrect

```

```

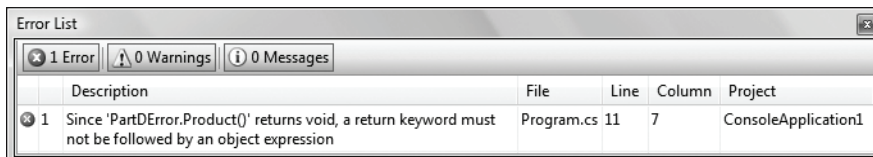
d) void Product()
{
    int a = 6, b = 5, c = 4, result;
    result = a * b * c;
    Console.WriteLine( "Result is " + result );
    return result;
}

```

```

1 // Exercise 7.5d Solution: PartDError.cs
2 using System;
3
4 public class PartDError
5 {
6     void Product()
7     {
8         int a = 6, b = 5, c = 4, result;
9         result = a * b * c;
10        Console.WriteLine( "Result is " + result );
11        return result;
12    } // end method Product
13 } // end class PartDError

```



ANS: Error: The method returns a value when it is not supposed to.
Correction: Change the return type from void to int.

```

1 // Exercise 7.5d Solution: PartDCorrect.cs
2 using System;
3
4 public class PartDCorrect
5 {
6     int Product()
7     {
8         int a = 6, b = 5, c = 4, result;
9         result = a * b * c;
10        Console.WriteLine( "Result is " + result );
11        return result;
12    } // end method Product
13 } // end class PartDCorrect

```

7.6 Write a complete C# application to prompt the user for the double radius of a sphere, and call method `SphereVolume` to calculate and display the volume of the sphere. Use the following statement to calculate the volume:

double volume = (4.0 / 3.0) * Math.PI * Math.Pow(radius, 3)

ANS:

```

1 // Exercise 7.6 Solution: Sphere.cs
2 // Calculate the volume of a sphere.
3 using System;
4
5 public class Sphere
6 {
7     // obtain radius from user and display volume of sphere
8     public static void Main( string[] args )
9     {
10        Console.Write( "Enter radius of sphere: " );
11        double radius = Convert.ToDouble( Console.ReadLine() );
12
13        Console.WriteLine( "Volume is {0:F3}", SphereVolume( radius ) );
14    } // end Main
15
16    // calculate and return sphere volume
17    public static double SphereVolume( double radius )
18    {
19        double volume = ( 4.0 / 3.0 ) * Math.PI * Math.Pow( radius, 3 );
20        return volume;
21    } // end method SphereVolume
22 } // end class Sphere

```

Enter radius of sphere: 4 Volume is 268.083
--

Exercises

7.7 What is the value of *x* after each of the following statements is executed?

a) `x = Math.Abs(7.5);`

ANS: 7.5

b) `x = Math.Floor(7.5);`

ANS: 7.0

c) `x = Math.Abs(0.0);`

ANS: 0.0

d) `x = Math.Ceiling(0.0);`

ANS: 0.0

e) `x = Math.Abs(-6.4);`

ANS: 6.4

f) `x = Math.Ceiling(-6.4);`

ANS: -6.0

g) `x = Math.Ceiling(-Math.Abs(-8 + Math.Floor(-5.5)));`

ANS: -14.0

7.8 A parking garage charges a \$2.00 minimum fee to park for up to three hours. The garage charges an additional \$0.50 per hour for each hour *or part thereof* in excess of three hours. The maximum charge for any given 24-hour period is \$10.00. Assume that no car parks for longer than 24 hours at a time. Write an application that calculates and displays the parking charges for each customer who parked in the garage yesterday. You should enter the hours parked for each customer. The application should display the charge for the current customer and should calculate and display the running total of yesterday's receipts. The application should use method `CalculateCharges` to determine the charge for each customer.

ANS:

```

1 // Exercise 7.8 Solution: Garage.cs
2 // Application calculates charges for parking
3 using System;
4
5 public class Garage
6 {
7     // begin calculating charges
8     public static void Main( string[] args )
9     {
10         decimal totalReceipts = 0M; // total fee collected for the day
11         decimal fee; // the charge for the current customer
12         double hours; // hours for the current customer
13
14         // read in the first customer's hours
15         Console.Write(
16             "Enter number of hours (a negative or zero to quit): " );
17         hours = Convert.ToDouble( Console.ReadLine() );
18
19         while ( hours > 0.0 )
20         {
21             // calculate and display the charges
22             fee = CalculateCharges( hours );
23             totalReceipts += fee;

```

```

24         Console.WriteLine(
25             "Current charge: {0:C}, Total receipts: {1:C}",
26             fee, totalReceipts );
27
28         // read in the next customer's hours
29         Console.Write(
30             "Enter number of hours (a negative or zero to quit): " );
31         hours = Convert.ToDouble( Console.ReadLine() );
32     } // end while
33 } // end Main
34
35 // determines fee based on time
36 public static decimal CalculateCharges( double hours )
37 {
38     // apply minimum charge
39     decimal charge = 2M;
40
41     // add extra fees as applicable
42     if ( hours > 3.0 )
43         charge = 2M + ( decimal ) ( 0.5 * Math.Ceiling( hours - 3.0 ) );
44
45     // apply maximum value if needed
46     if ( charge > 10M )
47         charge = 10M;
48
49     return charge;
50 } // end method CalculateCharges
51 } // end class Garage

```

```

Enter number of hours (a negative to quit): 5
Current charge: $3.00, Total receipts: $3.00
Enter number of hours (a negative to quit): 10
Current charge: $5.50, Total receipts: $8.50
Enter number of hours (a negative to quit): 24
Current charge: $10.00, Total receipts: $18.50
Enter number of hours (a negative to quit): -1

```

7.9 An application of method `Math.Floor` is rounding a value to the nearest integer. The statement

```
y = Math.Floor( x + 0.5 );
```

will round the number `x` to the nearest integer and assign the result to `y`. Write an application that reads `double` values and uses the preceding statement to round each of the numbers to the nearest integer. For each number processed, display both the original number and the rounded number.

ANS:

```

1 // Exercise 7.9 Solution: RoundingTest.cs
2 // Application tests Math.Floor.
3 using System;
4
5 public class RoundingTest
6 {

```

```

7  public static void Main( string[] args )
8  {
9      Console.WriteLine( "{0}\n{1}", "Enter decimal numbers.",
10                          "Type <Ctrl> z and press Enter to terminate input:" );
11
12      string line = Console.ReadLine();
13
14      while ( line != null )
15      {
16          double x = Convert.ToDouble( line );
17
18          Console.WriteLine( "Number: {0}\tMath.Floor( x + 0.5 ): {1}",
19                             x, Math.Floor( x + 0.5 ) );
20
21          line = Console.ReadLine();
22      } // end while
23  } // end Main
24 } // end class RoundingTest

```

```

Enter decimal numbers.
Type <Ctrl> z and press Enter to terminate input:
2.3
Number: 2.3      Math.Floor( x + 0.5 ): 2
5.5
Number: 5.5      Math.Floor( x + 0.5 ): 6
-1.3
Number: -1.3     Math.Floor( x + 0.5 ): -1
^Z

```

7.10 `Math.Floor` may be used to round a number to a specific decimal place. The statement

```
y = Math.Floor( x * 10 + 0.5 ) / 10;
```

rounds `x` to the tenths position (i.e., the first position to the right of the decimal point). The statement

```
y = Math.Floor( x * 100 + 0.5 ) / 100;
```

rounds `x` to the hundredths position (i.e., the second position to the right of the decimal point).

Write an application that defines four methods for rounding a number `x` in various ways:

- `RoundToInteger(number)`
- `RoundToTenths(number)`
- `RoundToHundredths(number)`
- `RoundToThousandths(number)`

For each value read, your application should display the original value, the number rounded to the nearest integer, the number rounded to the nearest tenth, the number rounded to the nearest hundredth and the number rounded to the nearest thousandth.

ANS:

```

1  // Exercise 7.10 Solution: Round.cs
2  // Application tests rounding with Math.Floor
3  using System;
4

```

```
5 public class Round
6 {
7     // displays the various Roundings for a number
8     public static void Main( string[] args )
9     {
10         Console.WriteLine( "{0}\n{1}", "Enter decimal numbers.",
11                             "Type <Ctrl> z and press Enter to terminate input:" );
12
13         string line = Console.ReadLine();
14
15         while ( line != null )
16         {
17             double x = Convert.ToDouble( line );
18
19             // display the various Roundings
20             Console.WriteLine( "The number: {0}", x );
21             Console.WriteLine( "Rounded to Integer: {0}",
22                                 RoundToInteger( x ) );
23             Console.WriteLine( "Rounded to Tenth: {0}",
24                                 RoundToTenths( x ) );
25             Console.WriteLine( "Rounded to Hundredth: {0}",
26                                 RoundToHundredths( x ) );
27             Console.WriteLine( "Rounded to Thousandth: {0}",
28                                 RoundToThousandths( x ) );
29
30             line = Console.ReadLine();
31         } // end while
32     } // end Main
33
34     // Round to ones place
35     public static double RoundToInteger( double number )
36     {
37         return ( Math.Floor( number + 0.5 ) );
38     } // end method RoundToInteger
39
40     // Round to tenths place
41     public static double RoundToTenths( double number )
42     {
43         return ( Math.Floor( number * 10 + 0.5 ) / 10 );
44     } // end method RoundToTenths
45
46     // Round to hundredths place
47     public static double RoundToHundredths( double number )
48     {
49         return ( Math.Floor( number * 100 + 0.5 ) / 100 );
50     } // end method RoundToHundredths
51
52     // Round to thousandths place
53     public static double RoundToThousandths( double number )
54     {
55         return ( Math.Floor( number * 1000 + 0.5 ) / 1000 );
56     } // end method RoundToThousandths
57 } // end class Round
```

```

Enter decimal numbers.
Type <Ctrl> z and press Enter to terminate input:
12.3456
The number: 12.3456
Rounded to Integer: 12
Rounded to Tenth: 12.3
Rounded to Hundredth: 12.35
Rounded to Thousandth: 12.346
80.386
The number: 80.386
Rounded to Integer: 80
Rounded to Tenth: 80.4
Rounded to Hundredth: 80.39
Rounded to Thousandth: 80.386
^Z

```

7.11 Answer each of the following questions:

a) What does it mean to choose numbers “at random”?

ANS: Every number has an equal chance of being chosen at any time.

b) Why is the Random class useful for simulating games of chance?

ANS: Because it produces a series of pseudorandom numbers.

c) Why is it often necessary to scale or shift the values produced by a Random object?

ANS: To produce random numbers in a specific range.

d) Why is computerized simulation of real-world situations a useful technique?

ANS: It enables more accurate predictions of random events, such as cars arriving at toll booths and people arriving in lines at a supermarket. The results of a simulation can help determine how many toll booths to have open or how many cashiers to have open at specified times.

7.12 Write statements that assign random integers to the variable *n* in the following ranges. Assume `Random randomNumbers = new Random()` has been defined and use the two-parameter version of the method `Random.Next`.

a) $1 \leq n \leq 2$

ANS: `n = randomNumbers.Next(1, 3);`

b) $1 \leq n \leq 100$

ANS: `n = randomNumbers.Next(1, 101);`

c) $0 \leq n \leq 9$

ANS: `n = randomNumbers.Next(0, 10);`

d) $1000 \leq n \leq 1112$

ANS: `n = randomNumbers.Next(1000, 1113);`

e) $-1 \leq n \leq 1$

ANS: `n = randomNumbers.Next(-1, 2);`

f) $-3 \leq n \leq 11$

ANS: `n = randomNumbers.Next(-3, 12);`

```

1 // Exercise 7.12 Solution: RandomRange.cs
2 using System;
3
4 public class RandomRange
5 {

```

```

6 public static void Main( string[] args )
7 {
8     Random randomNumbers = new Random();
9     int n;
10
11     // a)
12     n = randomNumbers.Next( 1, 3 );
13     Console.WriteLine( "randomNumbers.Next( 1, 3 ) = {0}", n );
14
15     // b)
16     n = randomNumbers.Next( 1, 101 );
17     Console.WriteLine( "randomNumbers.Next( 1, 101 ) = {0}", n );
18
19     // c)
20     n = randomNumbers.Next( 0, 10 );
21     Console.WriteLine( "randomNumbers.Next( 0, 10 ) = {0}", n );
22
23     // d)
24     n = randomNumbers.Next( 1000, 1113 );
25     Console.WriteLine( "randomNumbers.Next( 1000, 1113 ) = {0}", n );
26
27     // e)
28     n = randomNumbers.Next( -1, 2 );
29     Console.WriteLine( "randomNumbers.Next( -1, 2 ) = {0}", n );
30
31     // f)
32     n = randomNumbers.Next( -3, 12 );
33     Console.WriteLine( "randomNumbers.Next( -3, 12 ) = {0}", n );
34 } // end Main
35 } // end class RandomRange

```

```

randomNumbers.Next( 1, 3 ) = 2
randomNumbers.Next( 1, 101 ) = 43
randomNumbers.Next( 0, 10 ) = 1
randomNumbers.Next( 1000, 1113 ) = 1003
randomNumbers.Next( -1, 2 ) = 0
randomNumbers.Next( -3, 12 ) = 11

```

```

randomNumbers.Next( 1, 3 ) = 1
randomNumbers.Next( 1, 101 ) = 72
randomNumbers.Next( 0, 10 ) = 4
randomNumbers.Next( 1000, 1113 ) = 1044
randomNumbers.Next( -1, 2 ) = 0
randomNumbers.Next( -3, 12 ) = -2

```

7.13 For each of the following sets of integers, write a single statement that will display a number at random from the set. Assume `Random randomNumbers = new Random()` has been defined and use the one-parameter version of method `Random.Next`.

a) 2, 4, 6, 8, 10.

ANS: `Console.WriteLine(2 + randomNumbers.Next(5) * 2);`

b) 3, 5, 7, 9, 11.

ANS: `Console.WriteLine(3 + randomNumbers.Next(5) * 2);`

c) 6, 10, 14, 18, 22.

ANS: `Console.WriteLine(6 + randomNumbers.Next(5) * 4);`

```

1 // Exercise 7.13 Solution: RandomSet.cs
2 using System;
3
4 public class RandomSet
5 {
6     public static void Main( string[] args )
7     {
8         Random randomNumbers = new Random();
9
10        // a)
11        Console.WriteLine( 2 + randomNumbers.Next( 5 ) * 2 );
12
13        // b)
14        Console.WriteLine( 3 + randomNumbers.Next( 5 ) * 2 );
15
16        // c)
17        Console.WriteLine( 6 + randomNumbers.Next( 5 ) * 4 );
18    } // end Main
19 } // end class RandomSet

```

2
11
6

6
9
18

7.14 Write a method `IntegerPower(base, exponent)` that returns the value of $\text{base}^{\text{exponent}}$

For example, `IntegerPower(3, 4)` calculates 3^4 (or $3 * 3 * 3 * 3$). Assume that exponent is a positive integer and that base is an integer. Method `IntegerPower` should use a `for` or `while` loop to control the calculation. Do not use any `Math`-library methods. Incorporate this method into an application that reads integer values for base and exponent and performs the calculation with the `IntegerPower` method.

ANS:

```

1 // Exercise 7.14 Solution: Power.cs
2 // Application calculates an exponent
3 using System;
4
5 public class Power
6 {
7     // begin calculating integer powers
8     public static void Main( string[] args )
9     {

```

```

10     Console.Write( "Enter base: " );
11     int baseToRaise = Convert.ToInt32( Console.ReadLine() );
12
13     Console.Write( "Enter exponent (negative to quit): " );
14     int exponent = Convert.ToInt32( Console.ReadLine() );
15
16     // use a negative exponent as a sentinel
17     while ( exponent >= 0 )
18     {
19         Console.WriteLine( "{0} to the {1} is {2}",
20                             baseToRaise, exponent,
21                             IntegerPower( baseToRaise, exponent ) );
22
23         Console.Write( "Enter base: " );
24         baseToRaise = Convert.ToInt32( Console.ReadLine() );
25
26         Console.Write( "Enter exponent (negative to quit): " );
27         exponent = Convert.ToInt32( Console.ReadLine() );
28     } // end while
29 } // end Main
30
31 // raise integer baseToRaise to the exponent power
32 public static int IntegerPower( int baseToRaise, int exponent )
33 {
34     int product = 1;
35
36     for ( int x = 1; x <= exponent; x++ )
37         product *= baseToRaise;
38
39     return product;
40 } // end method IntegerPower
41 } // end class Power

```

```

Enter base: 2
Enter exponent (negative to quit): 6
2 to the 6 is 64
Enter base: 3
Enter exponent (negative to quit): 4
3 to the 4 is 81
Enter base: 5
Enter exponent (negative to quit): 5
5 to the 5 is 3125
Enter base: -1
Enter exponent (negative to quit): -1

```

7.15 Write method `Hypotenuse` that calculates the length of the hypotenuse of a right triangle when the lengths of the other two sides are given. The method should take two arguments of type `double` and return the hypotenuse as a `double`. Incorporate this method into an application that reads values for `side1` and `side2` and performs the calculation with the `Hypotenuse` method. Determine the length of the hypotenuse for each of the triangles in Fig. 7.26.

Triangle	Side 1	Side 2
1	3.0	4.0
2	5.0	12.0
3	8.0	15.0

Fig. 7.26 | Values for the sides of triangles in Exercise 7.15.

ANS:

```

1  // Exercise 7.15 Solution: Triangle.cs
2  // Application calculates the hypotenuse of a right triangle.
3  using System;
4
5  public class Triangle
6  {
7      // reads in two sides and outputs the hypotenuse
8      public static void Main( string[] args )
9      {
10         double side1; // first side of triangle
11         double side2; // second side of triangle
12
13         Console.Write( "Enter side 1 (negative to quit): " );
14         side1 = Convert.ToDouble( Console.ReadLine() );
15
16         while ( side1 > 0 )
17         {
18             Console.Write( "Enter side 2: " );
19             side2 = Convert.ToDouble( Console.ReadLine() );
20
21             Console.WriteLine( "Hypotenuse is: {0}",
22                               Hypotenuse( side1, side2 ) );
23
24             Console.Write( "Enter side 1 (negative to quit): " );
25             side1 = Convert.ToDouble( Console.ReadLine() );
26         } // end while
27     } // end Main
28
29     // calculate hypotenuse given lengths of two sides
30     public static double Hypotenuse( double side1, double side2 )
31     {
32         double hypotenuseSquared = Math.Pow( side1, 2 ) +
33             Math.Pow( side2, 2 );
34
35         return Math.Sqrt( hypotenuseSquared );
36     } // end method Hypotenuse
37 } // end class Triangle

```

```

Enter side 1 (negative to quit): 3
Enter side 2: 4
Hypotenuse is: 5
Enter side 1 (negative to quit): 5
Enter side 2: 12
Hypotenuse is: 13
Enter side 1 (negative to quit): 8
Enter side 2: 15
Hypotenuse is: 17
Enter side 1 (negative to quit): -1

```

7.16 Write method `Multiple` that determines, for a pair of integers, whether the second integer is a multiple of the first. The method should take two integer arguments and return `true` if the second is a multiple of the first and `false` otherwise. Incorporate this method into an application that inputs a series of pairs of integers (one pair at a time) and determines whether the second value in each pair is a multiple of the first.

ANS:

```

1  // Exercise 7.16 Solution: Multiplicity.cs
2  // Determines if the second number entered is a multiple of the first.
3  using System;
4
5  public class Multiplicity
6  {
7      // checks if the second number is a multiple of the first
8      public static void Main( string[] args )
9      {
10         int first; // the first number
11         int second; // the second number
12
13         Console.Write( "Enter first number (0 to exit): " );
14         first = Convert.ToInt32( Console.ReadLine() );
15
16         // use 0 as the sentinel value, since we cannot divide by zero
17         while ( first != 0 )
18         {
19             Console.Write( "Enter second number: " );
20             second = Convert.ToInt32( Console.ReadLine() );
21
22             if ( Multiple( first, second ) )
23                 Console.WriteLine( "{0} is a multiple of {1}",
24                                     second, first );
25             else
26                 Console.WriteLine( "{0} is not a multiple of {1}",
27                                     second, first );
28
29             Console.Write( "Enter first number (0 to exit): " );
30             first = Convert.ToInt32( Console.ReadLine() );
31         } // end while
32     } // end Main
33

```

```

34 // determine if first int is a multiple of the second
35 public static bool Multiple( int firstNumber, int secondNumber )
36 {
37     return secondNumber % firstNumber == 0;
38 } // end method Multiple
39 } // end class Multiplicity

```

```

Enter first number (0 to exit): 20
Enter second number: 30
30 is not a multiple of 20
Enter first number (0 to exit): 40
Enter second number: 20
20 is not a multiple of 40
Enter first number (0 to exit): 20
Enter second number: 40
40 is a multiple of 20
Enter first number (0 to exit): -1
Enter second number: 5
5 is a multiple of -1
Enter first number (0 to exit): 0

```

7.17 Write method `IsEven` that uses the remainder operator (%) to determine whether an integer is even. The method should take an integer argument and return `true` if the integer is even and `false` otherwise. Incorporate this method into an application that inputs a sequence of integers (one at a time) and determines whether each is even or odd.

ANS:

```

1 // Exercise 7.17 Solution: EvenOdd.cs
2 // Application determines whether a number is odd or even.
3 using System;
4
5 public class EvenOdd
6 {
7     // determines whether numbers are even or odd
8     public static void Main( string[] args )
9     {
10         Console.WriteLine( "{0}\n{1}",
11             "Enter numbers to determine whether they are even or odd.",
12             "Type <Ctrl> z and press Enter to terminate input:" );
13         string line = Console.ReadLine();
14
15         while ( line != null )
16         {
17             int number = Convert.ToInt32( line );
18
19             if ( IsEven( number ) )
20                 Console.WriteLine( "{0} is even", number );
21             else
22                 Console.WriteLine( "{0} is odd", number );
23
24             line = Console.ReadLine();
25         } // end while
26     } // end Main

```

```

27
28     // return true if number is even
29     public static bool IsEven( int number )
30     {
31         return number % 2 == 0;
32     } // end method IsEven
33 } // end class EvenOdd

```

Enter numbers to determine whether they are even or odd.
Type <Ctrl> z and press Enter to terminate input:

```

100
100 is even
20
20 is even
15
15 is odd
-1
-1 is odd
^Z

```

7.18 Write method `SquareOfAsterisks` that displays a solid square (the same number of rows and columns) of asterisks whose side length is specified in integer parameter `side`. For example, if `side` is 4, the method should display

```

****
****
****
****

```

Incorporate this method into an application that reads an integer value for `side` from the user and outputs the asterisks with the `SquareOfAsterisks` method.

ANS:

```

1  // Exercise 7.18 Solution: Square.cs
2  // Application draws a square of asterisks.
3  using System;
4
5  public class Square
6  {
7      // obtain value from user
8      public static void Main( string[] args )
9      {
10         Console.Write( "Enter square size: " );
11         int size = Convert.ToInt32( Console.ReadLine() );
12
13         SquareOfAsterisks( size );
14     } // end Main
15
16     // draw a square of asterisks
17     public static void SquareOfAsterisks( int side )
18     {
19         for ( int count = 1; count <= side * side; count++ )
20         {

```

```

21         Console.Write( "*" );
22
23         if ( count % side == 0 )
24             Console.WriteLine();
25     } // end for
26 } // end method SquareOfAsterisks
27 } // end class Square

```

Enter square size: 10

```

*****
*****
*****
*****
*****
*****
*****
*****
*****
*****

```

7.19 Modify the method created in Exercise 7.18 to form the square out of whatever character is contained in character parameter `FillCharacter`. Thus, if `side` is 5 and `FillCharacter` is "#", the method should display

```

#####
#####
#####
#####
#####

```

[Hint: Use the expression `Convert.ToChar(Console.Read())` to read a character from the user.]

ANS:

```

1  // Exercise 7.19 Solution: Square2.cs
2  // Application draws a square of an input character
3  using System;
4
5  public class Square2
6  {
7      // obtain value from user
8      public static void Main( string[] args )
9      {
10         Console.Write( "Enter square size: " );
11         int size = Convert.ToInt32( Console.ReadLine() );
12
13         Console.Write( "Enter square character: " );
14         char fillCharacter = Convert.ToChar( Console.Read() );
15
16         FillSquare( size, fillCharacter );
17     } // end Main
18
19     // draw a square of the passed character
20     public static void FillSquare( int side, char fillCharacter )
21     {

```

```

22     for ( int count = 1; count <= side * side; count++ )
23     {
24         Console.Write( fillCharacter );
25
26         if ( count % side == 0 )
27             Console.WriteLine();
28     } // end for
29 } // end method FillSquare
30 } // end class Square2

```

```

Enter square size: 10
Enter square character: a
aaaaaaaaaa
aaaaaaaaaa
aaaaaaaaaa
aaaaaaaaaa
aaaaaaaaaa
aaaaaaaaaa
aaaaaaaaaa
aaaaaaaaaa
aaaaaaaaaa
aaaaaaaaaa

```

7.20 Write an application that prompts the user for the radius of a circle and uses method `CircleArea` to calculate the area of the circle.

ANS:

```

1  // Exercise 7.20 Solution: Circle.cs
2  // Application calculates the area of a circle.
3  using System;
4
5  public class Circle
6  {
7      // calculate the areas of circles
8      public static void Main( string[] args )
9      {
10         Console.Write( "Enter the radius (negative to quit): " );
11         double radius = Convert.ToDouble( Console.ReadLine() );
12
13         while ( radius >= 0 )
14         {
15             Console.WriteLine( "Area is {0}", CircleArea( radius ) );
16
17             Console.Write( "Enter the radius (negative to quit): " );
18             radius = Convert.ToDouble( Console.ReadLine() );
19         } // end while
20     } // end Main
21
22     // calculate area
23     public static double CircleArea( double radius )
24     {
25         return Math.PI * radius * radius;
26     } // end method CircleArea
27 } // end class Circle

```

```

Enter the radius (negative to quit): 5
Area is 78.5398163397448
Enter the radius (negative to quit): 10
Area is 314.159265358979
Enter the radius (negative to quit): 1
Area is 3.14159265358979
Enter the radius (negative to quit): -1

```

7.21 Write code segments that accomplish each of the following tasks:

- a) Calculate the integer part of the quotient when integer a is divided by integer b.
- b) Calculate the integer remainder when integer a is divided by integer b.
- c) Use the application pieces developed in parts (a) and (b) to write a method `DisplayDigits` that receives an integer between 1 and 99999 and displays it as a sequence of digits, separating each pair of digits by two spaces. For example, the integer 4562 should appear as

```
4 5 6 2
```

- d) Incorporate the method developed in part (c) into an application that inputs an integer and calls `DisplayDigits` by passing the method the integer entered. Display the results.

ANS:

```

1 // Exercise 7.21 Solution: Digits.cs
2 // Application separates a five-digit number into its individual digits.
3 using System;
4
5 public class Digits
6 {
7     // displays the individual digits of a number
8     public static void Main( string[] args )
9     {
10         Console.Write( "Enter the integer (0 to exit): " );
11         int number = Convert.ToInt32( Console.ReadLine() );
12
13         while ( number != 0 )
14         {
15             if ( number <= 99999 && number >= 1 )
16                 DisplayDigits( number );
17             else
18                 Console.WriteLine( "number must be between 1 and 99999" );
19
20             Console.Write( "Enter the integer (0 to exit): " );
21             number = Convert.ToInt32( Console.ReadLine() );
22         } // end while
23     } // end Main
24
25     // part A
26     public static int Quotient( int a, int b )
27     {
28         return a / b;
29     } // end method Quotient
30

```

```

31 // part B
32 public static int Remainder( int a, int b )
33 {
34     return a % b;
35 } // end method Remainder
36
37 // part C
38 public static void DisplayDigits( int number )
39 {
40     int divisor = 1, digit;
41     string result = string.Empty;
42
43     // loop for highest divisor
44     for ( int i = 1; i < number; i *= 10 )
45         divisor = i;
46
47     while ( divisor >= 1 )
48     {
49         digit = Quotient( number, divisor );
50
51         result += ( digit + " " );
52
53         number = Remainder( number, divisor );
54         divisor = Quotient( divisor, 10 );
55     } // end while
56
57     Console.WriteLine( result );
58 } // end method DisplayDigits
59 } // end class Digits

```

```

Enter the integer (0 to exit): 34
3 4
Enter the integer (0 to exit): 567
5 6 7
Enter the integer (0 to exit): 8675309
number must be between 1 and 99999
Enter the integer (0 to exit): 99998
9 9 9 9 8
Enter the integer (0 to exit): 0

```

7.22 Implement the following integer methods:

- a) Method `Celsius` returns the Celsius equivalent of a Fahrenheit temperature, using the calculation

$$c = 5.0 / 9.0 * (f - 32);$$

- b) Method `Fahrenheit` returns the Fahrenheit equivalent of a Celsius temperature, using the calculation

$$f = 9.0 / 5.0 * c + 32;$$

- c) Use the methods from parts (a) and (b) to write an application that enables the user either to enter a Fahrenheit temperature and display the Celsius equivalent or to enter a Celsius temperature and display the Fahrenheit equivalent.

ANS:

```

1 // Exercise 7.22 Solution: Conversion.cs
2 // Application converts Fahrenheit to Celsius, and vice versa.
3 using System;
4
5 public class Conversion
6 {
7     // convert temperatures
8     public static void Main( string[] args )
9     {
10         int choice; // the user's choice in the menu
11
12         do
13         {
14             // display the menu
15             Console.WriteLine( "1. Fahrenheit to Celsius" );
16             Console.WriteLine( "2. Celsius to Fahrenheit" );
17             Console.WriteLine( "3. Exit" );
18             Console.Write( "Enter choice: " );
19             choice = Convert.ToInt32( Console.ReadLine() );
20
21             if ( ( choice == 1 ) || ( choice == 2 ) )
22             {
23                 Console.Write( "\nEnter {0} temperature: ",
24                     ( choice == 1 ) ? "Fahrenheit" : "Celsius" );
25                 int temperature = Convert.ToInt32( Console.ReadLine() );
26
27                 // convert the temperature appropriately
28                 switch ( choice )
29                 {
30                     case 1:
31                         Console.WriteLine( "{0} Fahrenheit is {1} Celsius",
32                             temperature, Celsius( temperature ) );
33                         break;
34                     case 2:
35                         Console.WriteLine( "{0} Celsius is {1} Fahrenheit",
36                             temperature, Fahrenheit( temperature ) );
37                         break;
38                 } // end switch
39
40                 Console.WriteLine(); // output a blank line
41             } // end if
42             else if ( choice != 3 ) // invalid input
43                 Console.WriteLine(
44                     "Invalid choice. Please choose again.\n" );
45
46         } while ( choice != 3 );
47     } // end Main
48
49     // return Celsius equivalent of Fahrenheit temperature
50     public static int Celsius( int fahrenheitTemperature )
51     {
52         return ( ( int ) ( 5.0 / 9.0 * ( fahrenheitTemperature - 32 ) ) );
53     } // end method Celsius

```

```

54
55     // return Fahrenheit equivalent of Celsius temperature
56     public static int Fahrenheit( int celsiusTemperature )
57     {
58         return ( ( int ) ( 9.0 / 5.0 * celsiusTemperature + 32 ) );
59     } // end method Fahrenheit
60 } // end class Conversion

```

```

1. Fahrenheit to Celsius
2. Celsius to Fahrenheit
3. Exit
Enter choice: 1

Enter Fahrenheit temperature: 212
212 Fahrenheit is 100 Celsius

1. Fahrenheit to Celsius
2. Celsius to Fahrenheit
3. Exit
Enter choice: 2

Enter Celsius temperature: 37
37 Celsius is 98 Fahrenheit

1. Fahrenheit to Celsius
2. Celsius to Fahrenheit
3. Exit
Enter choice: 3

```

7.23 Write a method `Minimum3` that returns the smallest of three floating-point numbers. Use the `Math.Min` method to implement `Minimum3`. Incorporate the method into an application that reads three values from the user, determines the smallest value and displays the result.

ANS:

```

1  // Exercise 7.23 Solution: Min.cs
2  // Application finds the minimum of three numbers
3  using System;
4
5  public class Min
6  {
7      // find the minimum of three numbers
8      public static void Main( string[] args )
9      {
10         double one; // first number
11         double two; // second number
12         double three; // third number
13
14         Console.WriteLine( "Type <Ctrl> z and press Enter to terminate" );
15         Console.Write( "Or enter first number: " );
16         string line = Console.ReadLine();
17
18         while ( line != null )
19         {

```

```

20         one = Convert.ToDouble( line );
21         Console.Write( "Enter second number: " );
22         two = Convert.ToDouble( Console.ReadLine() );
23         Console.Write( "Enter third number: " );
24         three = Convert.ToDouble( Console.ReadLine() );
25
26         Console.WriteLine( "Minimum is {0}",
27             Minimum3( one, two, three ) );
28
29         Console.WriteLine(
30             "Type <Ctrl> z and press Enter to terminate" );
31         Console.Write( "Or enter first number: " );
32         line = Console.ReadLine();
33     } // end while
34 } // end Main
35
36 // determine the smallest of three numbers
37 public static double Minimum3( double one, double two, double three )
38 {
39     // use a nested pair of Math.Min method calls
40     return Math.Min( Math.Min( one, two ), three );
41 } // end method Minimum3
42 } // end class Min

```

```

Type <Ctrl> z and press Enter to terminate
Or enter first number: 20
Enter second number: 30
Enter third number: 40
    Minimum is 20
Type <Ctrl> z and press Enter to terminate
Or enter first number: 50
Enter second number: 4
Enter third number: 1
    Minimum is 1
Type <Ctrl> z and press Enter to terminate
Or enter first number: 20
Enter second number: -1
Enter third number: 100
    Minimum is -1
Type <Ctrl> z and press Enter to terminate
Or enter first number: ^Z

```

7.24 (*Perfect Numbers*) An integer number is said to be a *perfect number* if its factors, including 1 (but not the number itself), sum to the number. For example, 6 is a perfect number, because $6 = 1 + 2 + 3$. Write method `Perfect` that determines whether parameter value is a perfect number. Use this method in an application that determines and displays all the perfect numbers between 2 and 1000. Display the factors of each perfect number to confirm that the number is indeed perfect.

ANS:

```

1 // Exercise 7.24 Solution: PerfectNumber.cs
2 // Application displays all perfect numbers between 2 and 1000.
3 using System;

```

```

4
5 public class PerfectNumber
6 {
7     // finds all the perfect numbers from 2 to 1000
8     public static void Main( string[] args )
9     {
10         for ( int number = 2; number <= 1000; number++ )
11         {
12             string result = Perfect( number );
13
14             if ( result != null )
15                 Console.WriteLine( "{0} is perfect.\n\tFactors: {1}",
16                                     number, result );
17         } // end for
18     } // end Main
19
20     // returns a string of factors if parameter is a
21     // perfect number, or null if it isn't.
22     public static string Perfect( int value )
23     {
24         int factorSum = 1;
25         string factors = "1 ";
26
27         for ( int test = 2; test <= value / 2; test++ )
28         {
29             if ( value % test == 0 )
30             {
31                 factorSum += test;
32                 factors += test + " ";
33             } // end if
34         } // end for
35
36         if ( factorSum == value )
37             return factors;
38
39         return null;
40     } // end method Perfect
41 } // end class PerfectNumber

```

```

6 is perfect.
    Factors: 1 2 3
28 is perfect.
    Factors: 1 2 4 7 14
496 is perfect.
    Factors: 1 2 4 8 16 31 62 124 248

```

7.25 An integer is said to be *prime* if it is greater than 1 and divisible by only 1 and itself. For example, 2, 3, 5 and 7 are prime, but 4, 6, 8 and 9 are not.

- Write a method that determines whether a number is prime.
- Use this method in an application that determines and displays all the prime numbers less than 10,000.

ANS:

```

1 // Exercise 7.25 Part A and B Solution: PrimeNum.cs
2 // Application calculates prime numbers
3 using System;
4
5 public class PrimeNum
6 {
7     // find the prime numbers between 2 and 10,000
8     public static void Main( string[] args )
9     {
10         Console.WriteLine( "Prime numbers between 2 and 10,000 are: " );
11
12         // test all numbers between 2 and 10000
13         for ( int m = 2; m <= 10000; m++ )
14             if ( Prime( m ) )
15                 Console.WriteLine( m );
16     } // end Main
17
18     // a helper method for determining whether a number is prime
19     // (This is the solution to 7.25, Part A.)
20     public static bool Prime( int n )
21     {
22         for ( int v = 2; v < n; v++ )
23             if ( n % v == 0 )
24                 return false;
25
26         return true;
27     } // end method Prime
28 } // end class PrimeNum

```

Prime numbers between 2 and 10,000 are:

```

2
3
5
7
11
13
17
19
23
...
9949
9967
9973

```

- c) Initially, you might think that $n/2$ is the upper limit for which you must test to see whether a number is prime, but you need only go as high as the square root of n . Rewrite the application, and run it both ways.

ANS:

```

1 // Exercise 7.25 Part C Solution: PrimeNum2.cs
2 // Application calculates prime numbers more efficiently
3 using System;
4
5 public class PrimeNum2
6 {
7     // find the prime numbers between 2 and 10,000
8     public static void Main( string[] args )
9     {
10         Console.WriteLine( "Prime numbers between 2 and 10,000 are: " );
11
12         // test all numbers between 2 and 10000
13         for ( int m = 2; m <= 10000; m++ )
14             if ( Prime( m ) )
15                 Console.WriteLine( m );
16     } // end Main
17
18     // a helper method for determining whether a number is prime
19     public static bool Prime( int n )
20     {
21         int max = ( int ) Math.Sqrt( n ); // the highest number to test
22
23         for ( int v = 2; v <= max; v++ )
24             if ( n % v == 0 )
25                 return false;
26
27         return true;
28     } // end method Prime
29 } // end class PrimeNum2

```

Prime numbers between 2 and 10,000 are:

```

2
3
5
7
11
13
17
19
23
...
9949
9967
9973

```

7.26 Write a method that takes an integer value and returns the number with its digits reversed. For example, given the number 7631, the method should return 1367. Incorporate the method into an application that reads a value from the user and displays the result.

ANS:

```

1 // Exercise 7.26 Solution: Reverse.cs
2 // Application takes a number and displays it with its digits reversed.
3 using System;
4
5 public class Reverse
6 {
7     // reverses an Integer
8     public static void Main( string[] args )
9     {
10         Console.Write( "Enter an integer (-1 to exit): " );
11         int number = Convert.ToInt32( Console.ReadLine() );
12
13         while ( number != -1 )
14         {
15             Console.WriteLine( "{0} reversed is {1}",
16                               number, ReverseDigits( number ) );
17
18             Console.Write( "Enter an integer (-1 to exit): " );
19             number = Convert.ToInt32( Console.ReadLine() );
20         } // end while
21     } // end Main
22
23     // display parameter number with digits reversed
24     public static int ReverseDigits( int number )
25     {
26         int reverseNumber = 0; // the number in reverse order
27         int placeValue; // the value at the current place
28
29         while ( number > 0 )
30         {
31             placeValue = number % 10;
32             number = number / 10;
33             reverseNumber = reverseNumber * 10 + placeValue;
34         } // end while
35
36         return reverseNumber;
37     } // end method ReverseDigits
38 } // end class Reverse

```

```

Enter an integer (-1 to exit): 100
100 reversed is 1
Enter an integer (-1 to exit): 8675309
8675309 reversed is 9035768
Enter an integer (-1 to exit): -1

```

7.27 (*Greatest Common Divisor*) The *greatest common divisor* (GCD) of two integers is the largest integer that evenly divides each of the two numbers. Write method `Gcd` that returns the greatest common divisor of two integers. Incorporate the method into an application that reads two values from the user and displays the result.

ANS:

```

1  // Exercise 7.27 Solution: Divisor.cs
2  // Application finds the greatest common divisor of two numbers.
3  using System;
4
5  public class Divisor
6  {
7      // finds the gcd of two numbers
8      public static void Main( string[] args )
9      {
10         int num1; // first number
11         int num2; // second number
12
13         Console.Write( "Enter first number (-1 to exit): " );
14         num1 = Convert.ToInt32( Console.ReadLine() );
15
16         while ( num1 != -1 )
17         {
18             Console.Write( "Enter second number: " );
19             num2 = Convert.ToInt32( Console.ReadLine() );
20
21             Console.WriteLine( "GCD is: {0}", Gcd( num1, num2 ) );
22
23             Console.Write( "Enter first number (-1 to exit): " );
24             num1 = Convert.ToInt32( Console.ReadLine() );
25         } // end while
26     } // end Main
27
28     // calculate the greatest common divisor using Euclid's algorithm--
29     // alternatively, you can simply check every number up to the
30     // lesser of x or y to see if it divides both x and y.
31     public static int Gcd( int x, int y )
32     {
33         int mod; // remainder of x / y
34
35         while ( y != 0 )
36         {
37             mod = x % y;
38             x = y;
39             y = mod;
40         } // end while
41
42         return x;
43     } // end method Gcd
44 } // end class Divisor

```

```

Enter first number (-1 to exit): 25
Enter second number: 10
GCD is: 5
Enter first number (-1 to exit): 49
Enter second number: 1
GCD is: 1
Enter first number (-1 to exit): 35
Enter second number: 48
GCD is: 1
Enter first number (-1 to exit): -1

```

7.28 Write method `QualityPoints` that inputs a student's average and returns 4 if the student's average is 90–100, 3 if the average is 80–89, 2 if the average is 70–79, 1 if the average is 60–69 and 0 if the average is lower than 60. Incorporate the method into an application that reads a value from the user and displays the result.

ANS:

```

1  // Exercise 7.28 Solution: Average.cs
2  // Application displays a number representing the student's average.
3  using System;
4
5  public class Average
6  {
7      public static void Main( string[] args )
8      {
9          Console.Write( "Enter average (-1 to quit): " );
10         int inputNumber = Convert.ToInt32( Console.ReadLine() );
11
12         while ( inputNumber != -1 )
13         {
14             if ( inputNumber >= 0 && inputNumber <= 100 )
15                 Console.WriteLine( "Point is: {0}",
16                                     QualityPoints( inputNumber ) );
17             else
18                 Console.WriteLine( "Invalid input." );
19
20             Console.Write( "Enter average (-1 to quit): " );
21             inputNumber = Convert.ToInt32( Console.ReadLine() );
22         } // end while
23     } // end Main
24
25     // return single-digit value of grade
26     public static int QualityPoints( int grade )
27     {
28         if ( grade >= 90 )
29             return 4;
30         else if ( grade >= 80 )
31             return 3;
32         else if ( grade >= 70 )
33             return 2;
34         else if ( grade >= 60 )
35             return 1;

```

```

36         else
37             return 0;
38     } // end method QualityPoints
39 } // end class Average

```

```

Enter average (-1 to quit): 95
Point is: 4
Enter average (-1 to quit): 64
Point is: 1
Enter average (-1 to quit): 88
Point is: 3
Enter average (-1 to quit): -1

```

7.29 (*Coin Tossing*) Write an application that simulates coin tossing. Let the application toss a coin each time the user chooses the “Toss Coin” menu option. Count the number of times each side of the coin appears. Display the results. The application should call a separate method `Flip` that takes no arguments and returns `false` for tails and `true` for heads. [Note: If the application realistically simulates coin tossing, each side of the coin should appear approximately half the time.]

ANS:

```

1  // Exercise 7.29 Solution: Coin.cs
2  // Application simulates tossing a coin.
3  using System;
4
5  public class Coin
6  {
7      private static Random randomNumbers = new Random();
8
9      // flips a coin many times
10     public static void Main( string[] args )
11     {
12         int heads = 0; // the number of times heads shows up
13         int tails = 0; // the number of times tails shows up
14         int choice; // the user's choice
15
16         do
17         {
18             // display a menu
19             Console.WriteLine( "1. Toss Coin" );
20             Console.WriteLine( "2. Exit" );
21             Console.Write( "Enter choice: " );
22             choice = Convert.ToInt32( Console.ReadLine() );
23
24             if ( choice == 1 )
25             {
26                 if ( Flip() )
27                     ++heads;
28                 else
29                     ++tails;
30
31                 Console.WriteLine( "Heads: {0}, Tails: {1}", heads, tails );
32             } // end if

```

```

33         else if ( choice != 2 ) // invalid input
34             Console.WriteLine( "Invalid choice. Please choose again." );
35
36     } while ( choice != 2 );
37 } // end Main
38
39 // simulate flipping
40 public static bool Flip()
41 {
42     return randomNumbers.Next( 2 ) == 1;
43 } // end method Flip
44 } // end class Coin

```

```

1. Toss Coin
2. Exit
Enter choice: 1
Heads: 0, Tails: 1
1. Toss Coin
2. Exit
Enter choice: 1
Heads: 0, Tails: 2
1. Toss Coin
2. Exit
Enter choice: 1
Heads: 1, Tails: 2
1. Toss Coin
2. Exit
Enter choice: 1
Heads: 2, Tails: 2
1. Toss Coin
2. Exit
Enter choice: 1
Heads: 2, Tails: 3
1. Toss Coin
2. Exit
Enter choice: 1
Heads: 3, Tails: 3
1. Toss Coin
2. Exit
Enter choice: 2

```

7.30 (*Guess the Number Game*) Write an application that plays “guess the number” as follows: Your application chooses the number to be guessed by selecting a random integer in the range 1 to 1000. The application displays the prompt *Guess a number between 1 and 1000*. The player inputs a first guess. If the player’s guess is incorrect, your application should display *Too high. Try again.* or *Too low. Try again.* to help the player “zero in” on the correct answer. The application should prompt the user for the next guess. When the user enters the correct answer, display *Congratulations. You guessed the number!* and allow the user to choose whether to play again. [Note: The guessing technique employed in this problem is similar to a binary search, which is discussed in Chapter 25.]

ANS:

```

1  // Exercise 7.30 Solution: Guess.cs
2  // Application plays guess the number.
3  using System;
4
5  public class Guess
6  {
7      static Random randomNumbers = new Random();
8      static int answer; // the answer to be guessed
9
10     // play games of guess the number
11     public static void Main( string[] args )
12     {
13         int userGuess; // the guess made by the user
14
15         NewGame();
16
17         Console.Write( "Guess (0 to exit): " );
18         userGuess = Convert.ToInt32( Console.ReadLine() );
19
20         while ( userGuess != 0 )
21         {
22             CheckUserGuess( userGuess );
23
24             Console.Write( "Guess (0 to exit): " );
25             userGuess = Convert.ToInt32( Console.ReadLine() );
26         } // end while
27     } // end Main
28
29     // create a new number to guess
30     public static int GetNumber()
31     {
32         return randomNumbers.Next( 1, 1001 );
33     } // end method GetNumber
34
35     // starts a new game
36     public static void NewGame()
37     {
38         answer = GetNumber();
39         Console.WriteLine( "Guess a number between 1 and 1000" );
40     } // end method NewGame
41
42     // judge user input, give feedback
43     public static void CheckUserGuess( int userGuess )
44     {
45         if ( userGuess < answer )
46             Console.WriteLine( "{0} is too low. Try again.", userGuess );
47         else if ( userGuess > answer )
48             Console.WriteLine( "{0} is too high. Try again.", userGuess );
49         else
50         {
51             Console.WriteLine(
52                 "Congratulations. You guessed the number!\n" );

```

```

53         // new game
54         NewGame();
55     } // end else
56 } // end method CheckUserGuess
57 } // end class Guess

```

```

Guess a number between 1 and 1000
Guess (0 to exit): 500
500 is too high. Try again.
Guess (0 to exit): 250
250 is too low. Try again.
Guess (0 to exit): 375
375 is too low. Try again.
Guess (0 to exit): 437
437 is too high. Try again.
Guess (0 to exit): 400
400 is too low. Try again.
Guess (0 to exit): 410
410 is too low. Try again.
Guess (0 to exit): 420
420 is too low. Try again.
Guess (0 to exit): 425
425 is too low. Try again.
Guess (0 to exit): 430
Congratulations. You guessed the number!

```

```

Guess a number between 1 and 1000
Guess (0 to exit): 0

```

7.31 Modify the application of Exercise 7.33 to count the number of guesses the player makes. If the number is 10 or fewer, display Either you know the secret or you got lucky! If the player guesses the number in 10 tries, display Aha! You know the secret! If the player makes more than 10 guesses, display You should be able to do better! Why should it take no more than 10 guesses? Well, with each “good guess,” the player should be able to eliminate half of the numbers. Now show why any number from 1 to 1000 can be guessed in 10 or fewer tries.

ANS: Guessing the middle number in the range of remaining numbers will always eliminate at least half the numbers (because the number guessed is eliminated as well). Repeatedly halving 1000 yields the values 500, 250, 125, 62, 31, 15, 7, 3, and 1. Because at most one element remains after halving 9 times, the user will always be able to guess by the 10th try.

```

1  // Exercise 7.31 Solution: Guess2.cs
2  // Application plays guess the number, counting the guesses.
3  using System;
4
5  public class Guess2
6  {
7      static Random randomNumbers = new Random();
8      static int answer; // the answer to be guessed
9      static int guesses; // the number of guesses the user has made
10

```

```
11 // play games of guess the number
12 public static void Main( string[] args )
13 {
14     int userGuess; // the guess made by the user
15
16     NewGame();
17
18     Console.Write( "Guess (0 to exit): " );
19     userGuess = Convert.ToInt32( Console.ReadLine() );
20
21     while ( userGuess != 0 )
22     {
23         ++guesses;
24         CheckUserGuess( userGuess );
25
26         Console.Write( "Guess (0 to exit): " );
27         userGuess = Convert.ToInt32( Console.ReadLine() );
28     } // end while
29 } // end Main
30
31 // create a new number to guess
32 public static int GetNumber()
33 {
34     return randomNumbers.Next( 1, 1001 );
35 } // end method GetNumber
36
37 // starts a new game
38 public static void NewGame()
39 {
40     guesses = 0;
41     answer = GetNumber();
42     Console.WriteLine( "Guess a number between 1 and 1000" );
43 } // end method NewGame
44
45 // judge user input, give feedback
46 public static void CheckUserGuess( int userGuess )
47 {
48     if ( userGuess < answer )
49         Console.WriteLine( "{0} is too low. Try again.", userGuess );
50     else if ( userGuess > answer )
51         Console.WriteLine( "{0} is too high. Try again.", userGuess );
52     else
53     {
54         DisplayMessage();
55         // new game
56         NewGame();
57     } // end else
58 } // end method CheckUserGuess
59
60 // display a message based on the number of tries
61 public static void DisplayMessage()
62 {
63     Console.WriteLine( "You guessed the number in {0} tries",
64         guesses );
```

```

65
66     if ( guesses < 10 )
67         Console.WriteLine(
68             "Either you know the secret or you got lucky!\n" );
69     else if ( guesses == 10 )
70         Console.WriteLine( "Ahah! You know the secret!\n" );
71     else
72         Console.WriteLine( "You should be able to do better!\n" );
73     } // end method DisplayMessage
74 } // end class Guess2

```

```

Guess a number between 1 and 1000
Guess (0 to exit): 500
500 is too low. Try again.
Guess (0 to exit): 750
750 is too low. Try again.
Guess (0 to exit): 875
875 is too low. Try again.
Guess (0 to exit): 950
950 is too low. Try again.
Guess (0 to exit): 980
980 is too high. Try again.
Guess (0 to exit): 965
965 is too high. Try again.
Guess (0 to exit): 958
958 is too high. Try again.
Guess (0 to exit): 954
954 is too low. Try again.
Guess (0 to exit): 956
956 is too low. Try again.
Guess (0 to exit): 955
955 is too low. Try again.
Guess (0 to exit): 957
You guessed the number in 11 tries
You should be able to do better!

```

```

Guess a number between 1 and 1000
Guess (0 to exit): 500
500 is too low. Try again.
Guess (0 to exit): 750
750 is too high. Try again.
Guess (0 to exit): 625
625 is too low. Try again.
Guess (0 to exit): 680
680 is too low. Try again.
Guess (0 to exit): 720
720 is too high. Try again.
Guess (0 to exit): 700
700 is too low. Try again.
Guess (0 to exit): 710
710 is too high. Try again.
Guess (0 to exit): 705
705 is too high. Try again.
Guess (0 to exit): 703
703 is too high. Try again.
Guess (0 to exit): 702

```

You guessed the number in 10 tries
Ahah! You know the secret!

Guess a number between 1 and 1000
Guess (0 to exit): 0

7.32 (*Distance Between Two Points*) Write method `Distance` to calculate the distance between two points $(x1, y1)$ and $(x2, y2)$. All numbers and return values should be of type `double`. Incorporate this method into an application that enables the user to enter the coordinates of the points.

ANS:

```

1  // Exercise 7.32 Solution: Points.cs
2  // Application calculates the distance between two points.
3  using System;
4
5  public class Points
6  {
7      // calculates the distance between two points
8      public static void Main( string[] args )
9      {
10         Console.WriteLine( "Press <Ctrl> z and press Enter to terminate" );
11         Console.Write( "Or Enter X1: " );
12         string line = Console.ReadLine();
13
14         // continually read in inputs until the user terminates
15         while ( line != null )
16         {
17             double x1 = Convert.ToDouble( line );
18             Console.Write( "Enter Y1: " );
19             double y1 = Convert.ToDouble( Console.ReadLine() );
20             Console.Write( "Enter X2: " );
21             double x2 = Convert.ToDouble( Console.ReadLine() );
22             Console.Write( "Enter Y2: " );
23             double y2 = Convert.ToDouble( Console.ReadLine() );
24
25             double distance = Distance( x1, y1, x2, y2 );
26             Console.WriteLine( "Distance is {0}\n", distance );
27
28             Console.WriteLine(
29                 "Press <Ctrl> z and press Enter to terminate" );
30             Console.Write( "Or Enter X1: " );
31             line = Console.ReadLine();
32         } // end while
33     } // end Main
34
35     // calculate distance between two points
36     public static double Distance( double x1, double y1,
37         double x2, double y2 )
38     {
39         return Math.Sqrt( Math.Pow( ( x1 - x2 ), 2 ) +
40             Math.Pow( ( y1 - y2 ), 2 ) );
41     } // end method Distance
42 } // end class Points

```

```

Press <Ctrl> z and press Enter to terminate
Or Enter X1: 4
Enter Y1: 0
Enter X2: 0
Enter Y2: 3
Distance is 5

```

```

Press <Ctrl> z and press Enter to terminate
Or Enter X1: ^Z

```

7.33 Modify the craps application of Fig. 7.9 to allow wagering. Initialize variable `balance` to 1000 dollars. Prompt the player to enter a wager. Check that wager is less than or equal to `balance`, and if it is not, have the user reenter wager until a valid wager is entered. After a correct wager is entered, run one game of craps. If the player wins, increase `balance` by wager and display the new balance. If the player loses, decrease `balance` by wager, display the new balance, check whether `balance` has become zero and, if so, display the message "Sorry. You busted!"

ANS:

```

1  // Exercise 7.33: Craps.cs
2  // Craps class simulates the dice game Craps.
3  // allows the user to place bets on games
4  using System;
5
6  public class Craps
7  {
8      // create random-number generator for use in method rollDice
9      private static Random randomNumbers = new Random();
10
11     static int balance; // the current balance
12     static int wager; // the current wager
13
14     // enumeration with constants that represent the game status
15     private enum Status { CONTINUE, WON, LOST };
16
17     // enumeration with constants that represent common rolls of the dice
18     private enum DiceNames
19     {
20         SNAKE_EYES = 2,
21         TREY = 3,
22         SEVEN = 7,
23         YO_LEVEN = 11,
24         BOX_CARS = 12
25     }
26
27     // allows the user to bet on games of Craps
28     public static void Main( string[] args )
29     {
30         balance = 1000; // start the user off with 1000
31
32         do
33         {

```

```

34         // prompt the user for a wager
35         Console.WriteLine( "Current balance is {0}\n", balance );
36         Console.Write( "Enter wager (-1 to quit): " );
37         wager = Convert.ToInt32( Console.ReadLine() );
38
39         if ( wager >= 0 )
40         {
41             if ( wager > balance )
42                 Console.WriteLine( "You don't have enough money!" );
43             else
44             {
45                 Play(); // play a game
46
47                 if ( balance <= 0 )
48                     Console.WriteLine( "Sorry you busted!" );
49             } // end else
50         } // end if
51         // terminate if the user quits or runs out of money
52     } while ( ( wager != -1 ) && ( balance > 0 ) );
53 } // end Main
54
55 // plays one game of craps
56 public static void Play()
57 {
58     int sumOfDice = 0; // sum of the dice
59     int myPoint = 0; // point if no win or loss on first roll
60
61     Status gameStatus; // can contain CONTINUE, WON or LOST
62
63     sumOfDice = RollDice(); // first roll of the dice
64
65     // determine game status and point based on sumOfDice
66     switch ( ( DiceNames ) sumOfDice )
67     {
68         case DiceNames.SEVEN: // win with 7 on first roll
69         case DiceNames.YO_LEVEN: // win with 11 on first roll
70             gameStatus = Status.WON;
71             break;
72         case DiceNames.SNAKE_EYES: // lose with 2 on first roll
73         case DiceNames.TREY: // lose with 3 on first roll
74         case DiceNames.BOX_CARS: // lose with 12 on first roll
75             gameStatus = Status.LOST;
76             break;
77         default: // did not win or lose, so remember point
78             gameStatus = Status.CONTINUE; // game is not over
79             myPoint = sumOfDice; // remember the point
80             Console.WriteLine( "Point is {0}", myPoint );
81             break;
82     } // end switch
83
84     // while game is not complete ...
85     while ( gameStatus == Status.CONTINUE )
86     {
87         sumOfDice = RollDice(); // roll dice again

```

```

88
89         // determine game status
90         if ( sumOfDice == myPoint ) // win by making point
91             gameStatus = Status.WON;
92         else
93             if ( sumOfDice == 7 ) // lose by rolling 7
94                 gameStatus = Status.LOST;
95     } // end while
96
97     // display won or lost message and change the balance
98     if ( gameStatus == Status.WON )
99     {
100         Console.WriteLine( "Player wins" );
101         balance += wager;
102     } // end if
103     else
104     {
105         Console.WriteLine( "Player loses" );
106         balance -= wager;
107     } // end else
108 } // end method Play
109
110 // roll dice, calculate sum and display results
111 public static int RollDice()
112 {
113     // pick random die values
114     int die1 = randomNumbers.Next( 1, 7 );
115     int die2 = randomNumbers.Next( 1, 7 );
116
117     int sum = die1 + die2; // sum die values
118
119     // display results of this roll
120     Console.WriteLine( "Player rolled {0} + {1} = {2}",
121         die1, die2, sum );
122
123     return sum; // return sum of dice
124 } // end method RollDice
125 } // end class Craps

```

Current balance is 1000

Enter wager (-1 to quit): 500

Player rolled 2 + 5 = 7

Player wins

Current balance is 1500

Enter wager (-1 to quit): 800

Player rolled 2 + 3 = 5

Point is 5

Player rolled 1 + 2 = 3

Player rolled 3 + 3 = 6

Player rolled 5 + 6 = 11

Player rolled 2 + 4 = 6

Player rolled 4 + 6 = 10

```

Player rolled 3 + 2 = 5
Player wins
Current balance is 2300

Enter wager (-1 to quit): 2000
Player rolled 4 + 1 = 5
Point is 5
Player rolled 3 + 6 = 9
Player rolled 5 + 6 = 11
Player rolled 6 + 3 = 9
Player rolled 1 + 3 = 4
Player rolled 5 + 4 = 9
Player rolled 6 + 1 = 7
Player loses
Current balance is 300

Enter wager (-1 to quit): 400
You don't have enough money!
Current balance is 300

Enter wager (-1 to quit): 300
Player rolled 3 + 1 = 4
Point is 4
Player rolled 1 + 4 = 5
Player rolled 1 + 4 = 5
Player rolled 1 + 2 = 3
Player rolled 4 + 1 = 5
Player rolled 5 + 1 = 6
Player rolled 3 + 3 = 6
Player rolled 2 + 4 = 6
Player rolled 2 + 3 = 5
Player rolled 2 + 4 = 6
Player rolled 3 + 4 = 7
Player loses
Sorry you busted!

```

7.34 Write an application that displays a table of the binary, octal, and hexadecimal equivalents of the decimal numbers in the range 1–256. If you are not familiar with these number systems, read Appendix C, Number Systems, first.

ANS:

```

1 // Exercise 7.34 Solution: NumberSystem.cs
2 // Converting a decimal number to binary, octal and hexadecimal.
3 using System;
4
5 public class NumberSystem
6 {
7     // displays conversions in binary, octal, and hexadecimal
8     public static void Main( string[] args )
9     {
10         Console.WriteLine( "{0,-8}{1,-12}{2,-8}{3,-8}",
11             "Decimal", "Binary", "Octal", "Hexadecimal" );
12

```

```
13 // display binary, octal and hexadecimal representation
14 // for each number
15 for ( int i = 1; i <= 256; i++ )
16 {
17     string binary = ConvertToBinary( i ); // binary representation
18     string octal = ConvertToOctal( i ); // octal representation
19     string hexadecimal = ConvertToHex( i ); // hex representation
20
21     Console.WriteLine( "{0,-8}{1,-12}{2,-8}{3,-8}",
22         i, binary, octal, hexadecimal );
23 } // end for
24 } // end Main
25
26 // returns a string representation of the decimal number in binary
27 public static string ConvertToBinary( int dec )
28 {
29     string binary = string.Empty;
30
31     while ( dec >= 1 )
32     {
33         int value = dec % 2;
34         binary = value + binary;
35         dec /= 2;
36     } // end binary while
37
38     return binary;
39 } // end method ConvertToBinary
40
41 // returns a string representation of the number in octal
42 public static string ConvertToOctal( int dec )
43 {
44     string octal = string.Empty;
45
46     // get octal representation
47     while ( dec >= 1 )
48     {
49         int value = dec % 8;
50         octal = value + octal;
51         dec /= 8;
52     } // end octal while
53
54     return octal;
55 } // end method ConvertToOctal
56
57 // returns a string representation of the number in hexadecimal
58 public static string ConvertToHex( int dec )
59 {
60     string hexadecimal = string.Empty;
61
62     // get hexadecimal representation
63     while ( dec >= 1 )
64     {
65         int value = dec % 16;
66
```

```

67         switch ( value )
68         {
69             case 10:
70                 hexadecimal = "A" + hexadecimal;
71                 break;
72             case 11:
73                 hexadecimal = "B" + hexadecimal;
74                 break;
75             case 12:
76                 hexadecimal = "C" + hexadecimal;
77                 break;
78             case 13:
79                 hexadecimal = "D" + hexadecimal;
80                 break;
81             case 14:
82                 hexadecimal = "E" + hexadecimal;
83                 break;
84             case 15:
85                 hexadecimal = "F" + hexadecimal;
86                 break;
87             default:
88                 hexadecimal = value + hexadecimal;
89                 break;
90         } // end switch
91
92         dec /= 16;
93     } // end hexadecimal while
94
95     return hexadecimal;
96 } // end method ConvertToHex
97 } // end class NumberSystem

```

Decimal	Binary	Octal	Hexadecimal
1	1	1	1
2	10	2	2
3	11	3	3
...			
254	11111110	376	FE
255	11111111	377	FF
256	100000000	400	100

7.35 Write recursive method `Power(base, exponent)` that, when called, returns $\text{base}^{\text{exponent}}$

For example, $\text{Power}(3, 4) = 3 * 3 * 3 * 3$. Assume that `exponent` is an integer greater than or equal to 1. [Hint: The recursion step should use the relationship

$$\text{base}^{\text{exponent}} = \text{base} * \text{base}^{\text{exponent} - 1}$$

and the terminating condition occurs when `exponent` is equal to 1, because

$$\text{base}^1 = \text{base}$$

Incorporate this method into an application that enables the user to enter the base and exponent.

ANS:

```

1 // Exercise 7.35 Solution: Exponential.cs
2 // Application calculates value of exponent.
3 using System;
4
5 public class Exponential
6 {
7     // call method Power passing user input
8     public static void Main( string[] args )
9     {
10         int baseNumber; // the base to raise to a power
11         int exponent; // the power to raise to
12
13         // prompt user for base and obtain value from user
14         Console.Write( "Enter base: " );
15         baseNumber = Convert.ToInt32( Console.ReadLine() );
16
17         // prompt user for exponent and obtain value from user
18         Console.Write( "Enter exponent: " );
19         exponent = Convert.ToInt32( Console.ReadLine() );
20
21         if ( exponent > 0 )
22         {
23             int result = Power( baseNumber, exponent );
24             Console.WriteLine( "Value is {0}", result );
25         } // end if
26         else
27             Console.WriteLine( "Invalid Exponent." );
28     } // end Main
29
30     // recursively calculate value of exponent.
31     public static int Power( int baseNumber, int exponent )
32     {
33         if ( exponent == 1 )
34             return baseNumber;
35         else
36             return baseNumber * Power( baseNumber, exponent - 1 );
37     } // end method Power
38 } // end class Exponential

```

```

Enter base: 2
Enter exponent: 8
Value is 256

```

```

Enter base: 5
Enter exponent: 3
Value is 125

```

7.36 (*Towers of Hanoi*) Every budding computer scientist must grapple with certain classic problems, and the *Towers of Hanoi* (see Fig. 7.27) is one of the most famous. Legend has it that in a tem-

ple in the Far East, priests are attempting to move a stack of disks from one peg to another. The initial stack has 64 disks threaded onto one peg and arranged from bottom to top by decreasing size. The priests are attempting to move the stack from this peg to a second peg under the constraints that exactly one disk is moved at a time and at no time may a larger disk be placed above a smaller disk. A third peg is available for temporarily holding disks. Supposedly, the world will end when the priests complete their task, so there is little incentive for us to facilitate their efforts.

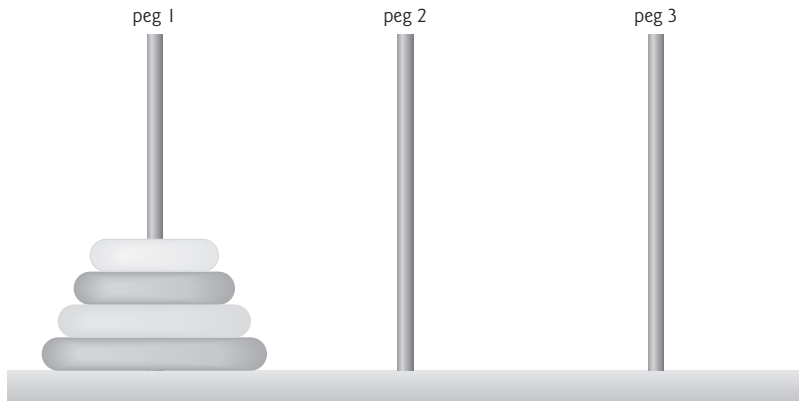


Fig. 7.27 | The Towers of Hanoi for the case with four disks.

Let us assume that the priests are attempting to move the disks from peg 1 to peg 3. We wish to develop an algorithm that will display the precise sequence of peg-to-peg disk transfers.

If we were to approach this problem with conventional methods, we would rapidly find ourselves hopelessly knotted up in managing the disks. Instead, if we attack the problem with recursion in mind, it immediately becomes tractable. Moving n disks can be viewed in terms of moving only $n - 1$ disks (hence the recursion) as follows:

- a) Move $n - 1$ disks from peg 1 to peg 2, using peg 3 as a temporary holding area.
- b) Move the last disk (the largest) from peg 1 to peg 3.
- c) Move the $n - 1$ disks from peg 2 to peg 3, using peg 1 as a temporary holding area.

The process ends when the last task involves moving $n = 1$ disk (i.e., the base case). This task is accomplished by simply moving the disk, without the need for a temporary holding area.

Write an application to solve the Towers of Hanoi problem. Allow the user to enter the number of disks. Use a recursive Tower method with four parameters:

- a) the number of disks to be moved,
- b) the peg on which these disks are initially threaded,
- c) the peg to which this stack of disks is to be moved, and
- d) the peg to be used as a temporary holding area.

Your application should display the precise instructions it will take to move the disks from the starting peg to the destination peg. For example, to move a stack of three disks from peg 1 to peg 3, your application should display the following series of moves:

```
1 --> 3 (This notation means "Move one disk from peg 1 to peg 3.")
1 --> 2
3 --> 2
1 --> 3
2 --> 1
```

2 --> 3
1 --> 3

ANS:

```

1  // Exercise 7.36 Solution: TowerOfHanoi.cs
2  // Application solves the towers of Hanoi problem, and
3  // demonstrates recursion
4  using System;
5
6  public class TowerOfHanoi
7  {
8      public static void Main( string[] args )
9      {
10         int disks; // number of disks in the tower
11
12         // prompt user for number of disks and read value
13         Console.Write( "Enter number of disks ( 1-9 ): " );
14         disks = Convert.ToInt32( Console.ReadLine() );
15
16         // actually sort the number of disks specified by user
17         Tower( disks, 1, 3, 2 );
18     } // end Main
19
20     // recursively move disks through towers
21     public static void Tower( int disks, int peg1, int peg3, int peg2 )
22     {
23         if ( disks == 1 )
24         {
25             Console.WriteLine( "{0} --> {1}", peg1, peg3 );
26             return;
27         }
28
29         // move ( disks - 1 ) disks from peg1 to peg2 recursively
30         Tower( disks - 1, peg1, peg2, peg3 );
31
32         // move last disk from peg1 to peg3 recursively
33         Console.WriteLine( "{0} --> {1}", peg1, peg3 );
34
35         // move ( disks - 1 ) disks from peg2 to peg3 recursively
36         Tower( disks - 1, peg2, peg3, peg1 );
37     } // end method Tower
38 } // end class TowerOfHanoi

```

```

Enter number of disks ( 1-9 ): 3
1 --> 3
1 --> 2
3 --> 2
1 --> 3
2 --> 1
2 --> 3
1 --> 3

```

```

Enter number of disks ( 1-9 ): 4
1 --> 2
1 --> 3
2 --> 3
1 --> 2
3 --> 1
3 --> 2
1 --> 2
1 --> 3
2 --> 3
2 --> 1
3 --> 1
2 --> 3
1 --> 2
1 --> 3
2 --> 3

```

7.37 What does the following method do?

```

// Parameter b must be a positive
// integer to prevent infinite recursion
public static int Mystery( int a, int b )
{
    if ( b == 1 )
        return a;
    else
        return a + Mystery( a, b - 1 );
}

```

ANS: The method returns the equivalent value of a times b where b is greater than 0.

```

1 // Exercise 7.37 Solution: MysteryTest.cs
2 // a method that returns the equivalent of a times b
3 // where b is greater than 0.
4 using System;
5
6 public class MysteryTest
7 {
8     public static void Main( string[] args )
9     {
10         Console.WriteLine( "Mystery( 5, 9 ) = {0}", Mystery( 5, 9 ) );
11         Console.WriteLine( "Mystery( 4, 8 ) = {0}", Mystery( 4, 8 ) );
12         Console.WriteLine( "Mystery( -6, 7 ) = {0}", Mystery( -6, 7 ) );
13     } // end Main
14
15     // Parameter b must be a positive
16     // integer to prevent infinite recursion
17     public static int Mystery( int a, int b )
18     {
19         if ( b == 1 )
20             return a;
21         else
22             return a + Mystery( a, b - 1 );
23     } // end method Mystery
24 } // end class MysteryTest

```

```
Mystery( 5, 9 ) = 45
Mystery( 4, 8 ) = 32
Mystery( -6, 7 ) = -42
```

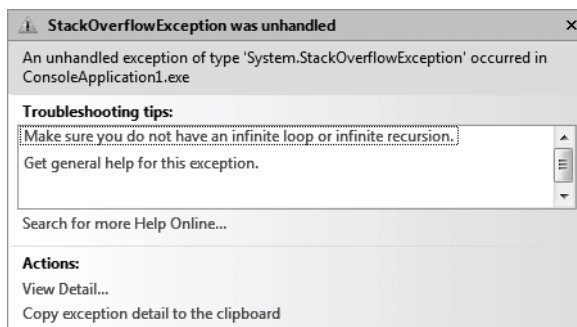
7.38 Find the error in the following recursive method, and explain how to correct it:

```
public static int Sum( int n )
{
    if ( n == 0 )
        return 0;
    else
        return n + Sum( n );
}
```

ANS: The method never reaches the base case. The recursion step should be:

```
return n + Sum( n - 1 );
```

```
1 // Exercise 7.38: SumTest.cs
2 using System;
3
4 public class SumTest
5 {
6     public static void Main( string[] args )
7     {
8         SumTest test = new SumTest();
9         Console.WriteLine( "Sum( 5 ) = {0}", test.Sum( 5 ) );
10    } // end Main
11
12    // causes an infinite loop
13    public static int Sum( int n )
14    {
15        if ( n == 0 )
16            return 0;
17        else
18            return n + Sum( n );
19    } // end method Sum
20 } // end class SumTest
```



```

1 // Exercise 7.38: SumCorrected.cs
2 using System;
3
4 public class SumCorrected
5 {
6     public static void Main( string[] args )
7     {
8         SumCorrected test = new SumCorrected();
9         Console.WriteLine( "Sum( 5 ) = {0}", test.Sum( 5 ) );
10        Console.WriteLine( "Sum( 8 ) = {0}", test.Sum( 8 ) );
11        Console.WriteLine( "Sum( 40 ) = {0}", test.Sum( 40 ) );
12    } // end Main
13
14    public static int Sum( int n )
15    {
16        if ( n == 0 )
17            return 0;
18        else
19            return n + Sum( n - 1 );
20    } // end method Sum
21 } // end class SumCorrected

```

```

Sum( 5 ) = 15
Sum( 8 ) = 36
Sum( 40 ) = 820

```

Making a Difference Exercises

As computer costs decline, it becomes feasible for every student, regardless of economic circumstance, to have a computer and use it in school. This creates exciting possibilities for improving the educational experience of all students worldwide as suggested by the next two exercises. [Note: Check out initiatives such as the One Laptop Per Child Project (www.1laptop.org). Also, research “green” laptops—and note the key “going green” characteristics of these devices. Look into the Electronic Product Environmental Assessment Tool (www.epeat.net) which can help you assess the “greenness” of desktops, notebooks and monitors to help you decide which products to purchase.]

7.39 (Computer-Assisted Instruction) The use of computers in education is referred to as *computer-assisted instruction (CAI)*. Write a program that will help an elementary school student learn multiplication. Use a Random object to produce two positive one-digit integers. The program should then prompt the user with a question, such as

How much is 6 times 7?

The student then inputs the answer. Next, the program checks the student’s answer. If it’s correct, display the message “Very good!” and ask another multiplication question. If the answer is wrong, display the message “No. Please try again.” and let the student try the same question repeatedly until the student finally gets it right. A separate function should be used to generate each new question. This function should be called once when the application begins execution and each time the user answers the question correctly.

ANS:

```
1 // Exercise 7.39 Solution: Multiply.cs
2 // Application generates single-digit multiplication problems
3 using System;
4
5 public class Multiply
6 {
7     static Random randomNumbers = new Random();
8
9     static int answer; // the correct answer
10
11     // ask the user to answer multiplication problems
12     public static void Main( string[] args )
13     {
14         int guess; // the user's guess
15
16         CreateQuestion(); // display the first question
17
18         Console.WriteLine( "Enter your answer (-1 to exit):" );
19         guess = Convert.ToInt32( Console.ReadLine() );
20
21         while ( guess != -1 )
22         {
23             CheckResponse( guess );
24
25             Console.WriteLine( "Enter your answer (-1 to exit):" );
26             guess = Convert.ToInt32( Console.ReadLine() );
27         } // end while
28     } // end method Quiz
29
30     // displays a new question and stores the corresponding answer
31     public static void CreateQuestion()
32     {
33         // get two random numbers between 0 and 9
34         int digit1 = randomNumbers.Next( 10 );
35         int digit2 = randomNumbers.Next( 10 );
36
37         answer = digit1 * digit2;
38         Console.WriteLine( "How much is {0} times {1}?",
39             digit1, digit2 );
40     } // end method CreateQuestion
41
42     // checks whether the user answered correctly
43     public static void CheckResponse( int guess )
44     {
45         if ( guess != answer )
46             Console.WriteLine( "No. Please try again." );
47         else
48         {
49             Console.WriteLine( "Very Good!" );
50             CreateQuestion();
51         } // end else
52     } // end method CheckResponse
53 } // end class Multiply
```

```

How much is 8 times 5?
Enter your answer (-1 to exit):
38
No. Please try again.
Enter your answer (-1 to exit):
40
Very Good!
How much is 7 times 2?
Enter your answer (-1 to exit):
14
Very Good!
How much is 6 times 8?
Enter your answer (-1 to exit):
-1

```

7.40 (*Computer-Assisted Instruction: Reducing Student Fatigue*) One problem in CAI environments is student fatigue. This can be reduced by varying the computer's responses to hold the student's attention. Modify the program of Exercise 7.39 so that various comments are displayed for each answer as follows:

Possible responses to a correct answer:

```

    Very good!
    Excellent!
    Nice work!
    Keep up the good work!

```

Possible responses to an incorrect answer:

```

    No. Please try again.
    Wrong. Try once more.
    Don't give up!
    No. Keep trying.

```

Use random-number generation to choose a number from 1 to 4 that will be used to select one of the four appropriate responses to each correct or incorrect answer. Use a `Select...Case` statement to issue the responses.

ANS:

```

1  // Exercise 7.40 Solution: Multiply2.cs
2  // Application generates single-digit multiplication problems
3  using System;
4
5  public class Multiply2
6  {
7      static Random randomNumbers = new Random();
8
9      static int answer; // the correct answer
10
11     // ask the user to answer multiplication problems
12     public static void Main( string[] args )
13     {
14         int guess; // the user's guess
15
16         CreateQuestion(); // display the first question
17

```

```
18 Console.WriteLine( "Enter your answer (-1 to exit):" );
19 guess = Convert.ToInt32( Console.ReadLine() );
20
21 while ( guess != -1 )
22 {
23     CheckResponse( guess );
24
25     Console.WriteLine( "Enter your answer (-1 to exit):" );
26     guess = Convert.ToInt32( Console.ReadLine() );
27 } // end while
28 } // end Main
29
30 // displays a new question and stores the corresponding answer
31 public static void CreateQuestion()
32 {
33     // get two random numbers between 0 and 9
34     int digit1 = randomNumbers.Next( 10 );
35     int digit2 = randomNumbers.Next( 10 );
36
37     answer = digit1 * digit2;
38     Console.WriteLine( "How much is {0} times {1}?",
39         digit1, digit2 );
40 } // end method CreateQuestion
41
42 // create a new response
43 public static string CreateResponse( bool correct )
44 {
45     string response = string.Empty;
46
47     if ( correct )
48     {
49         switch ( randomNumbers.Next( 4 ) )
50         {
51             case 0:
52                 response = "Very Good!";
53                 break;
54             case 1:
55                 response = "Excellent!";
56                 break;
57             case 2:
58                 response = "Nice work!";
59                 break;
60             case 3:
61                 response = "Keep up the good work!";
62                 break;
63         } // end switch
64     }
65     else
66     {
67         // otherwise, assume incorrect
68         switch ( randomNumbers.Next( 4 ) )
69         {
70             case 0:
71                 response = "No. Please try again.";
72                 break;
73             case 1:
74                 response = "Wrong. Try once more.";
75                 break;
```

```
73         case 2:
74             response = "Don't give up!";
75             break;
76         case 3:
77             response = "No. Keep trying.";
78             break;
79     } // end switch
80
81     return response;
82 } // end method CreateResponse
83
84 // checks whether the user answered correctly
85 public static void CheckResponse( int guess )
86 {
87     if ( guess != answer )
88         Console.WriteLine( CreateResponse( false ) );
89     else
90     {
91         Console.WriteLine( CreateResponse( true ) );
92         CreateQuestion();
93     } // end else
94 } // end method CheckResponse
95 } // end class Multiply2
```

```
How much is 4 times 4?
Enter your answer (-1 to exit):
16
Nice work!
How much is 0 times 5?
Enter your answer (-1 to exit):
5
No. Please try again.
Enter your answer (-1 to exit):
0
Nice work!
How much is 3 times 7?
Enter your answer (-1 to exit):
20
Wrong. Try once more.
Enter your answer (-1 to exit):
21
Very Good!
How much is 0 times 6?
Enter your answer (-1 to exit):
-1
```

Arrays: Solutions

8

Begin at the beginning, ... and go on till you come to the end: then stop.

—Lewis Carroll

Now go, write it before them in a table, and note it in a book.

—Isaiah 30:8

To go beyond is as wrong as to fall short.

—Confucius

Objectives

In this chapter you'll learn:

- To use arrays to store data in and retrieve data from lists and tables of values.
- To declare arrays, initialize arrays and refer to individual elements of arrays.
- To use **foreach** to iterate through arrays.
- To use implicitly typed local variables.
- To pass arrays to methods.
- To declare and manipulate multidimensional arrays.
- To write methods that use variable-length argument lists.
- To read command-line arguments into an application.

Self-Review Exercises

- 8.1** Fill in the blank(s) in each of the following statements:
- Lists and tables of values can be stored in _____.
ANS: arrays.
 - An array is a group of _____ (called elements) containing values that all have the same _____.
ANS: variables, type.
 - The _____ statement allows you to iterate through the elements in an array without using a counter.
ANS: `foreach`.
 - The number that refers to a particular array element is called the element's _____.
ANS: index (or position number).
 - An array that uses two indices is referred to as a(n) _____ array.
ANS: two-dimensional.
 - Use the `foreach` header _____ to iterate through double array numbers.
ANS: `foreach (double d in numbers)`.
 - Command-line arguments are stored in _____.
ANS: an array of strings, conventionally called `args`.
 - Use the expression _____ to receive the total number of arguments in a command line. Assume that command-line arguments are stored in `string[] args`.
ANS: `args.Length`.
 - Given the command `MyApplication test`, the first command-line argument is _____.
ANS: `test`.
 - A _____ in the parameter list of a method indicates that the method can receive a variable number of arguments.
ANS: `params` modifier.
- 8.2** Determine whether each of the following is *true* or *false*. If *false*, explain why.
- A single array can store values of many different types.
ANS: False. An array can store only values of the same type.
 - An array index should normally be of type `float`.
ANS: False. An array index must be an integer or an integer expression.
 - An individual array element that is passed to a method and modified in that method will contain the modified value when the called method completes execution.
ANS: For individual value-type elements of an array: False. A called method receives and manipulates a copy of the value of such an element, so modifications do not affect the original value. If the reference of an array is passed to a method, however, modifications to the array elements made in the called method are indeed reflected in the original. For individual elements of a reference type: True. A called method receives a copy of the reference of such an element, and changes to the referenced object will be reflected in the original array element.
 - Command-line arguments are separated by commas.
ANS: False. Command-line arguments are separated by white space.
- 8.3** Perform the following tasks for an array called `fractions`:
- Declare constant `ARRAY_SIZE` initialized to 10.
ANS: `const int ARRAY_SIZE = 10;`
 - Declare variable `fractions` which will reference an array with `ARRAY_SIZE` elements of type `double`. Initialize the elements to 0.
ANS: `double[] fractions = new double[ARRAY_SIZE];`

c) Name the element of the array with index 3.

ANS: `fractions[3]`

d) Assign the value 1.667 to the array element with index 9.

ANS: `fractions[9] = 1.667;`

e) Assign the value 3.333 to the array element with index 6.

ANS: `fractions[6] = 3.333;`

f) Sum all the elements of the array, using a for statement. Declare integer variable x as a control variable for the loop.

```
double total = 0.0;
for ( int x = 0; x < fractions.Length; x++ )
    total += fractions[ x ];
```

8.4 Perform the following tasks for an array called `table`:

a) Declare the variable and initialize it with a rectangular integer array that has three rows and three columns. Assume that constant `ARRAY_SIZE` has been declared to be 3.

ANS: `int[,] table = new int[ARRAY_SIZE, ARRAY_SIZE];`

b) How many elements does the array contain?

ANS: Nine.

c) Use a for statement to initialize each element of the array to the sum of its indices.

```
ANS: for ( int x = 0; x < table.GetLength( 0 ); x++ )
    for ( int y = 0; y < table.GetLength( 1 ); y++ )
        table[ x, y ] = x + y;
```

8.5 Find and correct the error in each of the following code segments:

a) `const int ARRAY_SIZE = 5;`

`ARRAY_SIZE = 10;`

ANS: Error: Assigning a value to a constant after it has been initialized.

Correction: Assign the correct value to the constant in the `const` declaration.

b) Assume `int[] b = new int[10];`

```
for ( int i = 0; i <= b.Length; i++ )
    b[ i ] = 1;
```

ANS: Error: Referencing an array element outside the bounds of the array (`b[10]`).

Correction: Change the `<=` operator to `<`.

c) Assume `int[,] a = { { 1, 2 }, { 3, 4 } };`

`a[1][1] = 5;`

ANS: Array indexing is performed incorrectly.

Correction: Change the statement to `a[1, 1] = 5;`.

Exercises

8.6 Fill in the blanks in each of the following statements:

a) One-dimensional array `p` contains four elements. The array access expressions for elements are _____, _____, _____ and _____.

ANS: `p[0]`, `p[1]`, `p[2]`, and `p[3]`

b) Naming an array's variable, stating its type and specifying the number of dimensions in the array is called _____ the array.

ANS: declaring

c) In a two-dimensional array, the first index identifies the _____ of an element and the second index identifies the _____ of an element.

ANS: row, column

d) An *m*-by-*n* array contains _____ rows, _____ columns and _____ elements.

ANS: *m*, *n*, *m* × *n*

4 Chapter 8 Arrays: Solutions

e) The name of the element in row 3 and column 5 of jagged array d is _____.

ANS: d[3][5]

8.7 Determine whether each of the following is *true* or *false*. If *false*, explain why.

a) To refer to a particular location or element within an array, we specify the name of the array's variable and the value of the particular element.

ANS: False. The name of the array's variable and the index are specified.

b) The declaration of a variable that references an array reserves memory for the array.

ANS: False. Memory for arrays must be dynamically allocated in C# with `new` or by specifying an initializer list to initialize the array's elements.

c) To indicate that 100 locations should be reserved for integer array p, the programmer writes the declaration

```
p[ 100 ];
```

ANS: False. The correct declaration is `int[] p = new int[100]`;

d) An application that initializes the elements of a 15-element array to 0 must contain at least one `for` statement.

ANS: False. Numeric arrays are automatically initialized to zero. Also, a member initializer list can be used, or other repetition statements could be used, or individual assignment statements could be used.

e) To total the elements of a two-dimensional array you must use nested `for` statements.

ANS: False. It is possible to total the elements of a two-dimensional array by adding all the elements in a single assignment statement or by using the other repetition statements (like `foreach` or `while`).

8.8 Write C# statements to accomplish each of the following tasks:

a) Display the value of the element of character array f with index 6.

ANS: `Console.Write(f[6]);`

b) Initialize each of the five elements of one-dimensional integer array g to 8.

ANS: `int[] g = { 8, 8, 8, 8, 8 };`

c) Total the 100 elements of floating-point array c.

ANS: `for (int k = 0; k < c.Length; k++)`

```
total += c[ k ];
```

d) Copy 11-element array a into the first portion of array b, which contains 34 elements.

ANS: `for (int j = 0; j < a.Length; j++)`

```
b[ j ] = a[ j ];
```

e) Determine and display the smallest and largest values contained in 99-element floating-point array w.

ANS: `double small = w[0];`

```
double large = w[ 0 ];
```

```
for ( int i = 0; i < w.Length; i++ )
```

```
if ( w[ i ] < small )
```

```
small = w[ i ];
```

```
else if ( w[ i ] > large )
```

```
large = w[ i ];
```

8.9 Consider the two-by-three rectangular integer array t.

a) Write a statement that declares t and creates the array.

ANS: `int[,] t = new int[2, 3];`

b) How many rows does t have?

ANS: two.

c) How many columns does t have?

ANS: three.

d) How many elements does `t` have?

ANS: six.

e) Write the access expressions for all the elements in row 1 of `t`.

ANS: `t[1, 0]`, `t[1, 1]`, `t[1, 2]`

f) Write the access expressions for all the elements in column 2 of `t`.

ANS: `t[0, 2]`, `t[1, 2]`

g) Write a single statement that sets the element of `t` in row 0 and column 1 to zero.

ANS: `t[0, 1] = 0;`

h) Write a sequence of statements that initializes each element of `t` to 1. Do not use a repetition statement.

ANS: `t[0, 0] = 1;`
`t[0, 1] = 1;`
`t[0, 2] = 1;`
`t[1, 0] = 1;`
`t[1, 1] = 1;`
`t[1, 2] = 1;`

i) Write a nested `for` statement that initializes each element of `t` to 3.

ANS: `for (int j = 0; j < t.GetLength(0); j++)`
`for (int k = 0; k < t.GetLength(1); k++)`
`t[j, k] = 3;`

j) Write a nested `for` statement that inputs values for the elements of `t` from the user.

ANS: `for (int j = 0; j < t.GetLength(0); j++)`
`for (int k = 0; k < t.GetLength(1); k++)`
`t[j, k] = Convert.ToInt32(Console.ReadLine());`

k) Write a sequence of statements that determines and displays the smallest value in `t`.

ANS: `int small = t[0, 0];`

```
for ( int j = 0; j < t.GetLength( 0 ); j++ )
    for ( int k = 0; k < t.GetLength( 1 ); k++ )
        if ( t[ j, k ] < small )
            small = t[ j, k ];
```

```
Console.WriteLine( small );
```

l) Write a statement that displays the elements of row 0 of `t`.

ANS: `Console.WriteLine("{0} {1} {2}", t[0, 0], t[0, 1], t[0, 2]);`

m) Write a statement that totals the elements of column 2 of `t`.

ANS: `int total = t[0, 2] + t[1, 2];`

n) Write a sequence of statements that displays the contents of `t` in tabular format. List the column indices as headings across the top, and list the row indices at the left of each row.

ANS: `Console.WriteLine("\t0\t1\t2\n");`

```
for ( int e = 0; e < t.GetLength( 0 ); e++ )
{
    Console.Write( e );
    for ( int r = 0; r < t.GetLength( 1 ); r++ )
        Console.Write( "\t{0}", t[ e, r ] );

    Console.WriteLine();
} // end for
```

8.10 (*Sales Commissions*) Use a one-dimensional array to solve the following problem: A company pays its salespeople on a commission basis. The salespeople receive \$200 per week plus 9% of their gross sales for that week. For example, a salesperson who grosses \$5000 in sales in a week receives \$200 plus 9% of \$5000, or a total of \$650. Write an application (using an array of counters) that determines how many of the salespeople earned salaries in each of the following ranges (assume that each salesperson's salary is truncated to an integer amount):

6 Chapter 8 Arrays: Solutions

- a) \$200–299
- b) \$300–399
- c) \$400–499
- d) \$500–599
- e) \$600–699
- f) \$700–799
- g) \$800–899
- h) \$900–999
- i) \$1000 and over

Summarize the results in tabular format.

ANS:

```
1 // Exercise 8.10 Solution: Sales.cs
2 // Application calculates the amount of pay for a salesperson and counts
3 // the number of salespeople that earned salaries in given ranges.
4 using System;
5
6 public class Sales
7 {
8     // counts the number of people in given salary ranges
9     public static void Main( string[] args )
10    {
11        int[] total = new int[ 9 ]; // totals for the various salaries
12
13        // read in values and assign them to the appropriate range
14        Console.Write( "Enter sales amount (negative to end): " );
15        decimal dollars = Convert.ToDecimal( Console.ReadLine() );
16
17        while ( dollars >= 0M )
18        {
19            decimal salary = dollars * 0.09M + 200M;
20            int range = ( int ) ( salary / 100M );
21
22            if ( range > 10 )
23                range = 10;
24
25            ++total[ range - 2 ];
26
27            Console.Write( "Enter sales amount (negative to end): " );
28            dollars = Convert.ToDecimal( Console.ReadLine() );
29        } // end while
30
31        // display chart
32        Console.WriteLine( "Range\t\tNumber" );
33
34        for ( int range = 0; range < total.Length - 1; range++ )
35            Console.WriteLine( "${0}-${1}\t{2}", ( 200 + 100 * range ),
36                               ( 299 + 100 * range ), total[ range ] );
37
```

```

38         // special case for the last range
39         Console.WriteLine( "$1000 and over\t{0}",
40             total[ total.Length - 1 ] );
41     } // end Main
42 } // end class Sales

```

```

Enter sales amount (negative to end): 2000
Enter sales amount (negative to end): 5000
Enter sales amount (negative to end): 4595
Enter sales amount (negative to end): 8250
Enter sales amount (negative to end): 1000
Enter sales amount (negative to end): 35060
Enter sales amount (negative to end): -1
Range           Number
$200-$299       1
$300-$399       1
$400-$499       0
$500-$599       0
$600-$699       2
$700-$799       0
$800-$899       0
$900-$999       1
$1000 and over  1

```

8.11 Write statements that perform the following one-dimensional-array operations:

a) Set the three elements of integer array counts to 0.

ANS: `for (int u = 0; u < counts.Length; u++)`
`counts[u] = 0;`

b) Add 1 to each of the four elements of integer array bonus.

ANS: `for (int v = 0; v < bonus.Length; v++)`
`++bonus[v];`

c) Display the five values of integer array bestScores in column format.

ANS: `for (int w = 0; w < bestScores.Length; w++)`
`Console.WriteLine("{0,6}{1,6}", w, bestScores[w]);`

```

1  // Exercise 8.11 Solution: Arrays.cs
2  using System;
3
4  public class Arrays
5  {
6      public static void Main( string[] args )
7      {
8          int[] counts = { 1, 2, 3 };
9          int[] bonus = { 4, 5, 6, 7 };
10         int[] bestScores = { 80, 85, 90, 95, 60 };
11
12         Console.WriteLine( "Array counts, before: " );
13         foreach ( int value in counts )
14             Console.Write( "{0} ", value );
15
16         // a)
17         for ( int u = 0; u < counts.Length; u++ )
18             counts[ u ] = 0;

```

```

19
20     Console.WriteLine( "\nArray counts, after: " );
21     foreach ( int value in counts )
22         Console.Write( "{0} ", value );
23
24     Console.WriteLine( "\n\nArray bonus, before: " );
25     foreach ( int value in bonus )
26         Console.Write( "{0} ", value );
27
28     // b)
29     for ( int v = 0; v < bonus.Length; v++ )
30         ++bonus[ v ];
31
32     Console.WriteLine( "\nArray bonus, after: " );
33     foreach ( int value in bonus )
34         Console.Write( "{0} ", value );
35
36     Console.WriteLine( "\n\n{0,6}{1,6}", "Index", "Value" );
37
38     // c)
39     for ( int w = 0; w < bestScores.Length; w++ )
40         Console.WriteLine( "{0,6}{1,6}", w, bestScores[ w ] );
41 } // end Main
42 } // end class Arrays

```

```

Array counts, before:
1 2 3
Array counts, after:
0 0 0

```

```

Array bonus, before:
4 5 6 7
Array bonus, after:
5 6 7 8

```

Index	Value
0	80
1	85
2	90
3	95
4	60

8.12 (*Duplicate Elimination*) Use a one-dimensional array to solve the following problem: Write an application that inputs five numbers, each of which is between 10 and 100, inclusive. As each number is read, display it only if it is not a duplicate of a number already read. Provide for the “worst case,” in which all five numbers are different. Use the smallest possible array to solve this problem. Display the complete set of unique values input after the user inputs each new value.

ANS:

```

1 // Exercise 8.12 Solution: Unique.cs
2 // Reads in 5 numbers and displays only the unique ones.
3 using System;

```

```
4
5 public class Unique
6 {
7     // gets 5 numbers from the user
8     public static void Main( string[] args )
9     {
10         int[] numbers = new int[ 4 ]; // list of unique numbers
11         int count = 0; // number of values read
12
13         while ( count < 5 )
14         {
15             Console.Write( "Enter number: " );
16             int number = Convert.ToInt32( Console.ReadLine() );
17
18             // validate the input
19             if ( number >= 10 && number <= 100 )
20             {
21                 // flags whether this number already exists
22                 bool containsNumber = false;
23
24                 // compare input number to unique numbers in array
25                 for ( int j = 0; j < count; j++ )
26                 {
27                     // if new number is duplicate, set the flag
28                     if ( number == numbers[ j ] )
29                         containsNumber = true;
30
31                     // display the list of valid unique numbers
32                     if ( numbers[ j ] > 0 )
33                         Console.Write( "{0} ", numbers[ j ] );
34                 } // end for
35
36                 // add only if the number is not there already
37                 if ( !containsNumber )
38                 {
39                     if ( count < numbers.Length )
40                         numbers[ count ] = number;
41                     Console.WriteLine( number ); // display new unique number
42                 } // end if
43                 else
44                 {
45                     Console.WriteLine();
46                     Console.WriteLine( "{0} has already been entered",
47                                         number );
48                 } // end else
49
50                 ++count; // increment count
51             } // end if
52             else
53                 Console.WriteLine( "number must be between 10 and 100" );
54         } // end while
55     } // end Main
56 } // end class Unique
```

```

Enter number: 50
50
Enter number: 20
50 20
Enter number: 50
50 20
50 has already been entered
Enter number: 80
50 20 80
Enter number: 101
number must be between 10 and 100
Enter number: 80
50 20 80
80 has already been entered

```

8.13 List the elements of three-by-five jagged array `sales` in the order in which they are set to 0 by the following code segment:

```

for ( int row = 0; row < sales.Length; row++ )
{
    for ( int col = 0; col < sales[row].Length; col++ )
    {
        sales[ row ][ col ] = 0;
    }
}

```

ANS: `sales[0][0]`, `sales[0][1]`, `sales[0][2]`, `sales[0][3]`,
`sales[0][4]`, `sales[1][0]`, `sales[1][1]`, `sales[1][2]`,
`sales[1][3]`, `sales[1][4]`, `sales[2][0]`, `sales[2][1]`,
`sales[2][2]`, `sales[2][3]`, `sales[2][4]`

8.14 Write an application that calculates the product of a series of integers that are passed to method `product` using a variable-length argument list. Test your method with several calls, each with a different number of arguments.

ANS:

```

1 // Exercise 8.14 Solution: ParamArrayTest.cs
2 // Using variable-length argument lists.
3 using System;
4
5 public class ParamArrayTest
6 {
7     // multiply numbers
8     public static int Product( params int[] numbers )
9     {
10         int product = 1;
11
12         // process variable-length argument list
13         foreach ( int number in numbers )
14             product *= number;
15
16         return product;
17     } // end method Product
18
19     public static void Main( string[] args )
20     {

```

```

21     // values to multiply
22     int a = 1;
23     int b = 2;
24     int c = 3;
25     int d = 4;
26     int e = 5;
27
28     // display integer values
29     Console.WriteLine( "a = {0}, b = {1}, c = {2}, d = {3}, e = {4}\n",
30         a, b, c, d, e );
31
32     // call product with different number of arguments in each call
33     Console.WriteLine( "The product of a and b is: {0}",
34         Product( a, b ) );
35     Console.WriteLine( "The product of a, b and c is: {0}",
36         Product( a, b, c ) );
37     Console.WriteLine( "The product of a, b, c and d is: {0}",
38         Product( a, b, c, d ) );
39     Console.WriteLine( "The product of a, b, c, d and e is: {0}",
40         Product( a, b, c, d, e ) );
41 } // end Main
42 } // end class ParamArrayTest

```

```

a = 1, b = 2, c = 3, d = 4, e = 5

The product of a and b is: 2
The product of a, b and c is: 6
The product of a, b, c and d is: 24
The product of a, b, c, d and e is: 120

```

8.15 Rewrite Fig. 8.2 so that the size of the array is specified by the first command-line argument. If no command-line argument is supplied, use 10 as the default size of the array.

ANS:

```

1  // Exercise 8.15 Solution: InitArray.cs
2  // Creating an array with size specified by the command-line argument.
3  using System;
4
5  public class InitArray
6  {
7      public static void Main( string[] args )
8      {
9          int[] array; // declare array
10         int size = 10; // default size of the array
11
12         // get size
13         if ( args.Length == 1 )
14             size = Convert.ToInt32( args[ 0 ] );
15
16         array = new int[ size ]; // create array with specified size
17
18         Console.WriteLine( "{0}{1,8}", "Index", "Value" );

```

```

19
20     // display array elements
21     for ( int count = 0; count < array.Length; count++ )
22         Console.WriteLine( "{0}{1,8}", count, array[ count ] );
23     } // end Main
24 } // end class InitArray

```

C:\Solutions\ch08\ex08_15>InitArray.exe 5

Index	Value
0	0
1	0
2	0
3	0
4	0

8.16 Write an application that uses a `foreach` statement to sum the double values passed by the command-line arguments. [*Hint*: Use static method `Convert.ToDouble` of class `Convert` to convert a string to a double value.]

ANS:

```

1 // Exercise 8.16 Solution: CalculateTotal.cs
2 // Calculates total of double values passed by command-line arguments.
3 using System;
4
5 public class CalculateTotal
6 {
7     public static void Main( string[] args )
8     {
9         double total = 0.0;
10
11         // calculate total
12         foreach ( string argument in args )
13             total += Convert.ToDouble( argument );
14
15         Console.WriteLine( "total is: {0:F}", total );
16     } // end Main
17 } // end class CalculateTotal

```

C:\Solutions\ch08\ex08_16>CalculateTotal.exe 1.11 2.22 3.33
total is: 6.66

8.17 (*Dice Rolling*) Write an application to simulate the rolling of two dice. The application should use an object of class `Random` once to roll the first die and again to roll the second die. The sum of the two values should then be calculated. Each die can show an integer value from 1 to 6, so the sum of the values will vary from 2 to 12, with 7 being the most frequent sum and 2 and 12 being the least frequent sums. Figure 8.29 shows the 36 possible combinations of the two dice. Your application should roll the dice 36,000 times. Use a one-dimensional array to tally the number of times each possible sum appears. Display the results in tabular format. Determine whether the totals are reasonable (e.g., there are six ways to roll a 7, so approximately one-sixth of the rolls should be 7).

	1	2	3	4	5	6
1	2	3	4	5	6	7
2	3	4	5	6	7	8
3	4	5	6	7	8	9
4	5	6	7	8	9	10
5	6	7	8	9	10	11
6	7	8	9	10	11	12

Fig. 8.24 | The 36 possible sums of two dice.

ANS:

```

1 // Exercise 8.17 Solution: Roll36.cs
2 // Application simulates rolling two six-sided dice 36,000 times.
3 using System;
4
5 public class Roll36
6 {
7     // simulate rolling of dice 36000 times
8     public static void Main( string[] args )
9     {
10         Random randomNumbers = new Random();
11
12         int face1; // number on first die
13         int face2; // number on second die
14         int[] totals = new int[ 13 ]; // frequencies of the sums
15
16         // roll the dice
17         for ( int roll = 1; roll <= 36000; roll++ )
18         {
19             face1 = randomNumbers.Next( 1, 7 );
20             face2 = randomNumbers.Next( 1, 7 );
21             ++totals[ face1 + face2 ];
22         } // end for
23
24         // display the table
25         Console.WriteLine( "{0,8}{1,12}{2,12}",
26             "Sum", "Frequency", "Percentage" );
27
28         // ignore indices 0 and 1
29         for ( int k = 2; k < totals.Length; k++ )
30         {
31             double percent = totals[ k ] / ( 360.0 );
32             Console.WriteLine( "{0,8}{1,12}{2,12:F}",
33                 k, totals[ k ], percent );
34         } // end for
35     } // end Main
36 } // end class Roll36

```

Sum	Frequency	Percentage
2	1032	2.87
3	1974	5.48
4	3088	8.58
5	3992	11.09
6	4936	13.71
7	5965	16.57
8	5034	13.98
9	4014	11.15
10	3013	8.37
11	1978	5.49
12	974	2.71

8.18 (*Game of Craps*) Write an application that runs 1000 games of craps (Fig. 7.9) and answers the following questions:

- How many games are won on the first roll, second roll, ..., twentieth roll and after the twentieth roll?
- How many games are lost on the first roll, second roll, ..., twentieth roll and after the twentieth roll?
- What are the chances of winning at craps? [*Note:* You should discover that craps is one of the fairest casino games.]
- What is the average length of a game of craps?

ANS:

```

1 // Exercise 8.18 Solution: Craps.cs
2 // Application plays 1000 games of craps and displays winning
3 // and losing statistics.
4 using System;
5
6 public class Craps
7 {
8     // create random number generator for use in method RollDice
9     private static Random randomNumbers = new Random();
10
11     // enumeration with constants that represent the game status
12     private enum Status { CONTINUE, WON, LOST };
13
14     // enumeration with constants that represent common rolls of the dice
15     private enum DiceNames
16     {
17         SNAKE_EYES = 2,
18         TREY = 3,
19         SEVEN = 7,
20         YO_LEVEN = 11,
21         BOX_CARS = 12
22     }
23
24     static int[] wins; // number of wins, by rolls
25     static int[] losses; // number of losses, by rolls
26     static int winSum = 0; // total number of wins
27     static int loseSum = 0; // total number of losses
28

```

```
29 // plays one game of craps
30 public static void Main( string[] args )
31 {
32     int sumOfDice = 0; // sum of the dice
33     int myPoint = 0; // point if no win or loss on first roll
34
35     Status gameStatus; // can contain CONTINUE, WON or LOST
36
37     int roll; // number of rolls for the current game
38
39     wins = new int[ 22 ]; // frequency of wins
40     losses = new int[ 22 ]; // frequency of losses
41
42     for ( int i = 1; i <= 1000; i++ )
43     {
44         sumOfDice = RollDice(); // first roll of the dice
45         roll = 1;
46
47         // determine game status and point based on sumOfDice
48         switch ( ( DiceNames ) sumOfDice )
49         {
50             case DiceNames.SEVEN: // win with 7 on first roll
51             case DiceNames.YO_LEVEN: // win with 11 on first roll
52                 gameStatus = Status.WON;
53                 break;
54             case DiceNames.SNAKE_EYES: // lose with 2 on first roll
55             case DiceNames.TREY: // lose with 3 on first roll
56             case DiceNames.BOX_CARS: // lose with 12 on first roll
57                 gameStatus = Status.LOST;
58                 break;
59             default: // did not win or lose, so remember point
60                 gameStatus = Status.CONTINUE; // game is not over
61                 myPoint = sumOfDice; // remember the point
62                 Console.WriteLine( "Point is {0}", myPoint );
63                 break;
64         } // end switch
65
66         // while game is not complete ...
67         while ( gameStatus == Status.CONTINUE )
68         {
69             sumOfDice = RollDice(); // roll dice again
70             ++roll;
71
72             // determine game status
73             if ( sumOfDice == myPoint ) // win by making point
74                 gameStatus = Status.WON;
75             else if ( sumOfDice == 7 ) // lose by rolling 7
76                 gameStatus = Status.LOST;
77         } // end while
78
79         // all roll results after 20th roll placed in last element
80         if ( roll > 21 )
81             roll = 21;
82     }
```

```

83         // increment number of wins in that roll
84         if ( gameStatus == Status.WON )
85         {
86             ++wins[ roll ];
87             ++winSum;
88         } // end if
89         else // increment number of losses in that roll
90         {
91             ++losses[ roll ];
92             ++loseSum;
93         } // end else
94     } // end for
95
96     PrintStats();
97 } // end Main
98
99 // display win/loss statistics
100 public static void PrintStats()
101 {
102     int totalGames = winSum + loseSum; // total number of games
103     int length = 0; // total length of the games
104
105     // display number of wins and losses on all rolls
106     for ( int i = 1; i <= 21; i++ )
107     {
108         if ( i == 21 )
109             Console.WriteLine( "{0} {1} {2} {3}",
110                               wins[ i ], "games won and", losses[ i ],
111                               "games lost on rolls after the 20th roll" );
112         else
113             Console.WriteLine( "{0} {1} {2} {3}{4}",
114                               wins[ i ], "games won and", losses[ i ],
115                               "games lost on roll #", i );
116
117         // for calculating length of game
118         // number of wins/losses on that roll multiplied
119         // by the roll number, then add them to length
120         length += wins[ i ] * i + losses[ i ] * i;
121     } // end for
122
123     // calculate chances of winning
124     Console.WriteLine( "\n{0} {1} / {2} = {3:F}%",
125                       "The chances of winning are", winSum, totalGames,
126                       ( 100.0 * winSum / totalGames ) );
127
128     Console.WriteLine( "The average game length is {0:F} rolls.",
129                       ( ( double ) length / totalGames ) );
130 } // end method PrintStats
131
132 // roll dice, calculate sum and display results
133 public static int RollDice()
134 {
135     // pick random die values
136     int die1 = randomNumbers.Next( 1, 7 );

```

```

137         int die2 = randomNumbers.Next( 1, 7 );
138         int sum = die1 + die2; // sum die values
139
140         return sum; // return sum of dice
141     } // end method RollDice
142 } // end class Craps

```

```

229 games won and 102 games lost on roll #1
72 games won and 121 games lost on roll #2
58 games won and 76 games lost on roll #3
36 games won and 57 games lost on roll #4
28 games won and 36 games lost on roll #5
23 games won and 32 games lost on roll #6
12 games won and 28 games lost on roll #7
13 games won and 16 games lost on roll #8
8 games won and 11 games lost on roll #9
7 games won and 8 games lost on roll #10
3 games won and 2 games lost on roll #11
1 games won and 4 games lost on roll #12
1 games won and 6 games lost on roll #13
2 games won and 3 games lost on roll #14
0 games won and 2 games lost on roll #15
0 games won and 0 games lost on roll #16
0 games won and 0 games lost on roll #17
0 games won and 0 games lost on roll #18
1 games won and 1 games lost on roll #19
0 games won and 0 games lost on roll #20
0 games won and 1 games lost on rolls after the 20th roll

```

The chances of winning are $494 / 1000 = 49.40\%$
The average game length is 3.34 rolls.

8.19 (*Airline Reservations System*) A small airline has just purchased a computer for its new automated reservations system. You have been asked to develop the new system. You are to write an application to assign seats on each flight of the airline's only plane (capacity: 10 seats).

Your application should display the following alternatives: Please type 1 for First Class and Please type 2 for Economy. If the user types 1, your application should assign a seat in the first-class section (seats 1–5). If the user types 2, your application should assign a seat in the economy section (seats 6–10).

Use a one-dimensional array of simple type `bool` to represent the seating chart of the plane. Initialize all the elements of the array to `false` to indicate that all the seats are empty. As each seat is assigned, set the corresponding element of the array to `true` to indicate that the seat is no longer available.

Your application should never assign a seat that has already been assigned. When the economy section is full, your application should ask the person if it is acceptable to be placed in the first-class section (and vice versa). If yes, make the appropriate seat assignment. If no, display the message "Next flight leaves in 3 hours."

ANS: NO SOLUTION PROVIDED

8.20 (*Total Sales*) Use a rectangular array to solve the following problem: A company has three salespeople (1 to 3) who sell five different products (1 to 5). Once a day, each salesperson passes in a slip for each type of product sold. Each slip contains the following:

a) The salesperson number

- b) The product number
- c) The total dollar value of that product sold that day

Thus, each salesperson passes in between 0 and 5 sales slips per day. Assume that the information from all of the slips for last month is available. Write an application that will read all the information for last month's sales and summarize the total sales by salesperson and by product. All totals should be stored in rectangular array sales. After processing all the information for last month, display the results in tabular format, with each column representing a particular salesperson and each row representing a particular product. Cross-total each row to get the total sales of each product for last month. Cross-total each column to get the total sales by salesperson for last month. Your tabular output should include these cross-totals to the right of the totaled rows and below the totaled columns.

ANS:

```

1  // Exercise 8.20 Solution: Sales2.cs
2  // Application totals sales for salespeople and products.
3  using System;
4
5  public class Sales2
6  {
7      public static void Main( string[] args )
8      {
9          // sales array holds data on number of each product sold
10         // by each salesman
11         decimal[ , ] sales = new decimal[ 5, 3 ];
12
13         Console.Write( "Enter sales person number (-1 to end): " );
14         int person = Convert.ToInt32( Console.ReadLine() );
15
16         while ( person != -1 )
17         {
18             Console.Write( "Enter product number: " );
19             int product = Convert.ToInt32( Console.ReadLine() );
20             Console.Write( "Enter sales amount: " );
21             decimal amount = Convert.ToDecimal( Console.ReadLine() );
22
23             // error-check the input
24             if ( person >= 1 && person < 4 &&
25                 product >= 1 && product < 6 && amount >= 0 )
26                 sales[ product - 1, person - 1 ] += amount;
27             else
28                 Console.WriteLine( "Invalid input!" );
29
30             Console.Write( "Enter sales person number (-1 to end): " );
31             person = Convert.ToInt32( Console.ReadLine() );
32         } // end while
33
34         // total for each salesperson
35         decimal[] salesPersonTotal = new decimal[ 4 ];
36
37         // display the table
38         Console.WriteLine( "\n{0,10}{1,16}{2,16}{3,16}{4,12}",
39             "Product", "Salesperson 1", "Salesperson 2",
40             "Salesperson 3", "Total" );

```

```

41
42     // for each column of each row, display the appropriate
43     // value representing a person's sales of a product
44     for ( int row = 0; row < 5; row++ )
45     {
46         decimal productTotal = 0M;
47         Console.Write( "{0,10}", ( row + 1 ) );
48
49         for ( int column = 0; column < 3; column++ )
50         {
51             Console.Write( "{0,16:C}", sales[ row, column ] );
52             productTotal += sales[ row, column ];
53             salesPersonTotal[ column ] += sales[ row, column ];
54         } // end for
55
56         Console.WriteLine( "{0,12:C}", productTotal );
57     } // end for
58
59     Console.Write( "{0,10}", "Total" );
60
61     for ( int column = 0; column < 3; column++ )
62         Console.Write( "{0,16:C}", salesPersonTotal[ column ] );
63
64     Console.WriteLine();
65 } // end Main
66 } // end class Sales2

```

```

Enter sales person number (-1 to end): 1
Enter product number: 1
Enter sales amount: 100
Enter sales person number (-1 to end): 1
Enter product number: 4
Enter sales amount: 500
Enter sales person number (-1 to end): 2
Enter product number: 2
Enter sales amount: 300
Enter sales person number (-1 to end): 4
Enter product number: 2
Enter sales amount: 1000
Invalid input!
Enter sales person number (-1 to end): 3
Enter product number: 2
Enter sales amount: 1000
Enter sales person number (-1 to end): -1

```

Product	Salesperson 1	Salesperson 2	Salesperson 3	Total
1	\$100.00	\$0.00	\$0.00	\$100.00
2	\$0.00	\$300.00	\$1,000.00	\$1,300.00
3	\$0.00	\$0.00	\$0.00	\$0.00
4	\$500.00	\$0.00	\$0.00	\$500.00
5	\$0.00	\$0.00	\$0.00	\$0.00
Total	\$600.00	\$300.00	\$1,000.00	

8.21 (*Turtle Graphics*) The Logo language made the concept of *turtle graphics* famous. Imagine a mechanical turtle that walks around the room under the control of a C# application. The turtle holds a pen in one of two positions—up or down. While the pen is down, the turtle traces out shapes as it moves, and while the pen is up, the turtle moves about freely without writing anything. In this problem, you'll simulate the operation of the turtle and create a computerized sketchpad.

Use 20-by-20 rectangular array `floor` that is initialized to 0. Read commands from an array that contains them. Keep track at all times of the current position of the turtle and whether the pen is currently up or down. Assume that the turtle always starts at position (0, 0) of the floor with its pen up. The set of turtle commands your application must process are shown in Fig. 8.30.

Command	Meaning
1	Pen up
2	Pen down
3	Turn right
4	Turn left
5,10	Move forward 10 spaces (replace 10 for a different number of spaces)
6	Display the 20-by-20 array
9	End of data (sentinel)

Fig. 8.25 | Turtle graphics commands.

Suppose that the turtle is somewhere near the center of the floor. The following “application” would draw and display a 12-by-12 square, leaving the pen in the up position:

```

2
5,12
3
5,12
3
5,12
3
5,12
1
6
9

```

As the turtle moves with the pen down, set the appropriate elements of array `floor` to 1s. When the 6 command (display the array) is given, wherever there is a 1 in the array, display an asterisk or any character you choose. Wherever there is a 0, display a blank.

Write an application to implement the turtle graphics capabilities discussed here. Write several turtle graphics applications to draw interesting shapes. Add other commands to increase the power of your turtle graphics language.

ANS:

```

1 // Exercise 8.21: TurtleGraphics.cs
2 // Drawing turtle graphics based on turtle commands.
3 using System;
4

```

```

5 public class TurtleGraphics
6 {
7     const int MAXCOMMANDS = 100; // maximum size of command array
8     const int SIZE = 20; // size of the drawing area
9
10    static int[ , ] floor; // array representing the floor
11    static int[ , ] commandArray; // list of commands
12
13    static int count; // the current number of commands
14    static int xPos; // the x Position of the turtle
15    static int yPos; // the y Position of the turtle
16
17    // enters the commands for the turtle graphics
18    public static void Main( string[] args )
19    {
20        count = 0;
21        commandArray = new int[ MAXCOMMANDS, 2 ];
22        floor = new int[ SIZE, SIZE ];
23
24        Console.Write( "Enter command (9 to end input): " );
25        int inputCommand = Convert.ToInt32( Console.ReadLine() );
26
27        while ( inputCommand != 9 && count < MAXCOMMANDS )
28        {
29            commandArray[ count, 0 ] = inputCommand;
30
31            // prompt for forward spaces
32            if ( inputCommand == 5 )
33            {
34                Console.Write( "Enter forward spaces: " );
35                commandArray[ count, 1 ] =
36                    Convert.ToInt32( Console.ReadLine() );
37            } // end if
38
39            ++count;
40
41            Console.Write( "Enter command (9 to end input): " );
42            inputCommand = Convert.ToInt32( Console.ReadLine() );
43        } // end while
44
45        ExecuteCommands();
46    } // end Main
47
48    // executes the commands in the command array
49    public static void ExecuteCommands()
50    {
51        int commandNumber = 0; // the current position in the array
52        int direction = 0; // the direction the turtle is facing
53        int distance = 0; // the distance the turtle will travel
54        int command; // the current command
55        bool penDown = false; // whether the pen is up or down
56        xPos = 0;
57        yPos = 0;
58    }

```

```

59     command = commandArray[ commandNumber, 0 ];
60
61     // continue executing commands until either reach the end
62     // or reach the max commands
63     while ( commandNumber < count )
64     {
65
66         // determine what command was entered
67         // and perform desired action
68         switch ( command )
69         {
70             case 1: // pen up
71                 penDown = false;
72                 break;
73             case 2: // pen down
74                 penDown = true;
75                 break;
76             case 3: // turn right
77                 direction = TurnRight( direction );
78                 break;
79             case 4: // turn left
80                 direction = TurnLeft( direction );
81                 break;
82             case 5: // move
83                 distance = commandArray[ commandNumber, 1 ];
84                 MovePen( penDown, floor, direction, distance );
85                 break;
86             case 6: // display the drawing
87                 Console.WriteLine( "\nThe drawing is:\n" );
88                 PrintArray( floor );
89                 break;
90         } // end switch
91
92         command = commandArray[ ++commandNumber, 0 ];
93     } // end while
94 } // end method ExecuteCommands
95
96 // method to turn turtle to the right
97 public static int TurnRight( int d )
98 {
99     return ++d > 3 ? 0 : d;
100 } // end method TurnRight
101
102 // method to turn turtle to the left
103 public static int TurnLeft( int d )
104 {
105     return --d < 0 ? 3 : d;
106 } // end method TurnLeft
107
108 // method to move the pen
109 public static void MovePen( bool down, int[ , ] floor, int dir,
110     int dist )
111 {
112     int j; // looping variable

```

```

113
114 // determine which way to move pen
115 switch ( dir )
116 {
117     case 0: // move to right
118         for ( j = 1; j <= dist && yPos + j < SIZE; j++ )
119             if ( down )
120                 floor[ xPos, yPos + j ] = 1;
121
122         yPos += j - 1;
123         break;
124
125     case 1: // move down
126         for ( j = 1; j <= dist && xPos + j < SIZE; j++ )
127             if ( down )
128                 floor[ xPos + j, yPos ] = 1;
129
130         xPos += j - 1;
131         break;
132
133     case 2: // move to left
134         for ( j = 1; j <= dist && yPos - j >= 0; j++ )
135             if ( down )
136                 floor[ xPos, yPos - j ] = 1;
137
138         yPos -= j - 1;
139         break;
140
141     case 3: // move up
142         for ( j = 1; j <= dist && xPos - j >= 0; j++ )
143             if ( down )
144                 floor[ xPos - j, yPos ] = 1;
145
146         xPos -= j - 1;
147         break;
148 } // end switch
149 } // end method MovePen
150
151 // method to display array drawing
152 public static void PrintArray( int[ , ] floor )
153 {
154     // display array
155     for ( int i = 0; i < SIZE; i++ )
156     {
157         for ( int j = 0; j < SIZE; j++ )
158             Console.Write( ( floor[ i, j ] == 1 ? "*" : " " ) );
159
160         Console.WriteLine();
161     } // end for
162 } // end method PrintArray
163 } // end class TurtleGraphics

```

```

Enter command (9 to end input): 2
Enter command (9 to end input): 5
Enter forward spaces: 12
Enter command (9 to end input): 3
Enter command (9 to end input): 5
Enter forward spaces: 12
Enter command (9 to end input): 3
Enter command (9 to end input): 5
Enter forward spaces: 12
Enter command (9 to end input): 3
Enter command (9 to end input): 5
Enter forward spaces: 12
Enter command (9 to end input): 1
Enter command (9 to end input): 6
Enter command (9 to end input): 9

```

The drawing is:

```

*****
*           *
*           *
*           *
*           *
*           *
*           *
*           *
*           *
*           *
*           *
*****

```

8.22 (*Knight's Tour*) One of the more interesting puzzles for chess buffs is the Knight's Tour problem, originally proposed by the mathematician Euler. Can the chess piece called the knight move around an empty chessboard and touch each of the 64 squares once and only once? We study this intriguing problem in depth here.

The knight makes only L-shaped moves (two spaces in one direction and one space in a perpendicular direction). Thus, as shown in Fig. 8.31, from a square near the middle of an empty chessboard, the knight (labeled K) can make eight different moves (numbered 0 through 7).

- a) Draw an eight-by-eight chessboard on a sheet of paper, and attempt a Knight's Tour by hand. Put a 1 in the starting square, a 2 in the second square, a 3 in the third and so on. Before starting the tour, estimate how far you think you'll get, remembering that a full tour consists of 64 moves. How far did you get? Was this close to your estimate?
- b) Now let us develop an application that will move the knight around a chessboard. The board is represented by eight-by-eight rectangular array board. Each square is initialized to zero. We describe each of the eight possible moves in terms of their horizontal and vertical components. For example, a move of type 0, as shown in Fig. 8.31, consists of moving two squares horizontally to the right and one square vertically upward. A move of type 2 consists of moving one square horizontally to the left and two squares vertically upward. Horizontal moves to the left and vertical moves upward are indicated with negative numbers. The eight moves may be described by two one-dimensional arrays, `horizontal` and `vertical`, as follows:

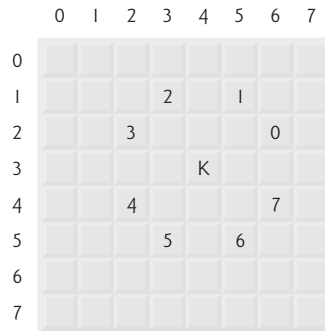


Fig. 8.26 | The eight possible moves of the knight.

```
horizontal[ 0 ] = 2      vertical[ 0 ] = -1
horizontal[ 1 ] = 1      vertical[ 1 ] = -2
horizontal[ 2 ] = -1     vertical[ 2 ] = -2
horizontal[ 3 ] = -2     vertical[ 3 ] = -1
horizontal[ 4 ] = -2     vertical[ 4 ] = 1
horizontal[ 5 ] = -1     vertical[ 5 ] = 2
horizontal[ 6 ] = 1      vertical[ 6 ] = 2
horizontal[ 7 ] = 2      vertical[ 7 ] = 1
```

Let variables `currentRow` and `currentColumn` indicate the row and column, respectively, of the knight's current position. To make a move of type `moveNumber`, where `moveNumber` is between 0 and 7, your application should use the statements

```
currentRow += vertical[ moveNumber ];
currentColumn += horizontal[ moveNumber ];
```

Write an application to move the knight around the chessboard. Keep a counter that varies from 1 to 64. Record the latest count in each square the knight moves to. Test each potential move to see if the knight has already visited that square. Test every potential move to ensure that the knight does not land off the chessboard. Run the application. How many moves did the knight make?

ANS:

```
1 // Exercise 8.22 Part B Solution: Knight1.cs
2 // Knight's Tour
3 using System;
4
5 public class Knight1
6 {
7     static Random randomNumbers = new Random();
8
9     static int[ , ] board; // gameboard
10
11     // moves
12     static int[] horizontal = { 2, 1, -1, -2, -2, -1, 1, 2 };
13     static int[] vertical = { -1, -2, -2, -1, 1, 2, 2, 1 };
14 }
```

```
15 // runs a tour
16 public static void Main( string[] args )
17 {
18     int currentRow; // the row position on the chessboard
19     int currentColumn; // the column position on the chessboard
20     int moveNumber = 0; // the current move number
21
22     board = new int[ 8, 8 ]; // gameboard
23
24     int testRow; // row position of next possible move
25     int testColumn; // column position of next possible move
26
27     // randomize initial board position
28     currentRow = randomNumbers.Next( 8 );
29     currentColumn = randomNumbers.Next( 8 );
30
31     board[ currentRow, currentColumn ] = ++moveNumber;
32     bool done = false;
33
34     // continue until knight can no longer move
35     while ( !done )
36     {
37         bool goodMove = false;
38
39         // check all possible moves until we find one that's legal
40         for ( int moveType = 0; moveType < 8 && !goodMove;
41             moveType++ )
42         {
43             testRow = currentRow + vertical[ moveType ];
44             testColumn = currentColumn + horizontal[ moveType ];
45             goodMove = ValidMove( testRow, testColumn );
46
47             // test if new move is valid
48             if ( goodMove )
49             {
50                 currentRow = testRow;
51                 currentColumn = testColumn;
52                 board[ currentRow, currentColumn ] = ++moveNumber;
53             } // end if
54         } // end for
55
56         // if no valid moves, knight can no longer move
57         if ( !goodMove )
58             done = true;
59         // if 64 moves have been made, a full tour is complete
60         else if ( moveNumber == 64 )
61             done = true;
62     } // end while
63
64     Console.WriteLine( "The tour ended with {0} moves.", moveNumber );
65
66     if ( moveNumber == 64 )
67         Console.WriteLine( "This was a full tour!" );
68     else
69         Console.WriteLine( "This was not a full tour." );
```

```

70
71     PrintTour();
72 } // end Main
73
74 // checks for valid move
75 public static bool ValidMove( int row, int column )
76 {
77     // returns false if the move is off the chessboard, or if
78     // the knight has already visited that position
79     // NOTE: This test stops as soon as it becomes false
80     return ( row >= 0 && row < 8 && column >= 0 && column < 8
81             && board[ row, column ] == 0 );
82 } // end method ValidMove
83
84 // display Knight's tour path
85 public static void PrintTour()
86 {
87     Console.Write( "\n " );
88
89     // display numbers for column
90     for ( int k = 0; k < 8; k++ )
91         Console.Write( "{0,3}", k );
92
93     Console.WriteLine( "\n" );
94
95     for ( int row = 0; row < board.GetLength( 0 ); row++ )
96     {
97         Console.Write( "{0,-2}", row );
98
99         for ( int column = 0; column < board.GetLength( 1 ); column++ )
100             Console.Write( "{0,3}", board[ row, column ] );
101
102         Console.WriteLine();
103     } // end for
104 } // end method PrintTour
105 } // end class Knight1

```

The tour ended with 32 moves.
This was not a full tour.

	0	1	2	3	4	5	6	7
0	32	13	18	3	16	11	8	1
1	19	4	15	12	7	2	27	10
2	14	31	6	17	28	9	24	0
3	5	20	29	0	23	0	0	26
4	30	0	22	0	0	25	0	0
5	21	0	0	0	0	0	0	0
6	0	0	0	0	0	0	0	0
7	0	0	0	0	0	0	0	0

- c) After attempting to write and run a Knight's Tour application, you have probably developed some valuable insights. We'll use these insights to develop a *heuristic* for moving the knight. Heuristics do not guarantee success, but a carefully developed heuristic

greatly improves the chance of success. You may have observed that the outer squares are more troublesome than the squares nearer the center of the board. In fact, the most troublesome and inaccessible squares are the four corners.

Intuition may suggest that you should attempt to move the knight to the most troublesome squares first and leave open those that are easiest to get to, so that when the board gets congested near the end of the tour, there will be a greater chance of success.

We could develop an “accessibility heuristic” by classifying each of the squares according to how accessible it is and always moving the knight (using the knight’s L-shaped moves) to the most inaccessible square. We label two-dimensional array accessibility with numbers indicating from how many squares each particular square is accessible. On a blank chessboard, each of the 16 squares nearest the center is rated as 8, each corner square is rated as 2, and the other squares have accessibility numbers of 3, 4 or 6 as follows:

```

2 3 4 4 4 4 3 2
3 4 6 6 6 6 4 3
4 6 8 8 8 8 6 4
4 6 8 8 8 8 6 4
4 6 8 8 8 8 6 4
4 6 8 8 8 8 6 4
3 4 6 6 6 6 4 3
2 3 4 4 4 4 3 2

```

Write a new version of the Knight’s Tour, using the accessibility heuristic. The knight should always move to the square with the lowest accessibility number. In case of a tie, the knight may move to any of the tied squares. Therefore, the tour may begin in any of the four corners. [Note: As the knight moves around the chessboard and more squares become occupied, your application should reduce the accessibility numbers. In this way, at any given time during the tour, each available square’s accessibility number will remain equal to precisely the number of squares from which that square may be reached.] Run this version of your application. Did you get a full tour? Modify the application to run 64 tours, one starting from each square of the chessboard. How many full tours did you get?

ANS:

```

1 // Exercise 8.22 Part C Solution: Knight2.cs
2 // Knight's Tour - heuristic version
3 using System;
4
5 public class Knight2
6 {
7     static Random randomNumbers = new Random();
8
9     static int[ , ] access = { { 2, 3, 4, 4, 4, 4, 3, 2 },
10                                { 3, 4, 6, 6, 6, 6, 4, 3 },
11                                { 4, 6, 8, 8, 8, 8, 6, 4 },
12                                { 4, 6, 8, 8, 8, 8, 6, 4 },
13                                { 4, 6, 8, 8, 8, 8, 6, 4 },
14                                { 4, 6, 8, 8, 8, 8, 6, 4 },
15                                { 3, 4, 6, 6, 6, 6, 4, 3 },
16                                { 2, 3, 4, 4, 4, 4, 3, 2 } };
17

```

```

18  static int[ , ] board; // gameboard
19  static int accessNumber; // the current access number
20
21  // moves
22  static int[] horizontal = { 2, 1, -1, -2, -2, -1, 1, 2 };
23  static int[] vertical = { -1, -2, -2, -1, 1, 2, 2, 1 };
24
25  // create a board and attempt a tour
26  public static void Main( string[] args )
27  {
28      int currentRow; // the row position on the chessboard
29      int currentColumn; // the column position on the chessboard
30      int moveNumber = 0; // the current move number
31
32      int testRow; // row position of next possible move
33      int testColumn; // column position of next possible move
34      int minRow = -1; // row position of move with minimum access
35      int minColumn = -1; // row position of move with minimum access
36
37      board = new int[ 8, 8 ];
38
39      // randomize initial board position
40      currentRow = randomNumbers.Next( 8 );
41      currentColumn = randomNumbers.Next( 8 );
42
43      board[ currentRow, currentColumn ] = ++moveNumber;
44      bool done = false;
45
46      // continue touring until finished traversing
47      while ( !done )
48      {
49          accessNumber = 99;
50
51          // try all possible moves
52          for ( int moveType = 0; moveType < board.GetLength( 0 );
53              moveType++ )
54          {
55              // new position of hypothetical moves
56              testRow = currentRow + vertical[ moveType ];
57              testColumn = currentColumn + horizontal[ moveType ];
58
59              if ( ValidMove( testRow, testColumn ) )
60              {
61                  // obtain access number
62                  if ( access[ testRow, testColumn ] < accessNumber )
63                  {
64                      // if this is the lowest access number thus far,
65                      // then set this move to be our next move
66                      accessNumber = access[ testRow, testColumn ];
67
68                      minRow = testRow;
69                      minColumn = testColumn;
70                  } // end if
71

```

```

72         // position access number tried
73         --access[ testRow, testColumn ];
74     } // end if
75 } // end for
76
77     // traversing done
78     if ( accessNumber == 99 ) // no valid moves
79         done = true;
80     else
81     { // make move
82         currentRow = minRow;
83         currentColumn = minColumn;
84         board[ currentRow, currentColumn ] = ++moveNumber;
85     } // end else
86 } // end while
87
88 Console.WriteLine( "The tour ended with {0} moves.", moveNumber );
89
90 if ( moveNumber == 64 )
91     Console.WriteLine( " This was a full tour!" );
92 else
93     Console.WriteLine( " This was not a full tour." );
94
95 PrintTour();
96 } // end Main
97
98 // checks for valid move
99 public static bool ValidMove( int row, int column )
100 {
101     // returns false if the move is off the chessboard, or if
102     // the knight has already visited that position
103     // NOTE: This test stops as soon as it becomes false
104     return ( row >= 0 && row < 8 && column >= 0 && column < 8
105             && board[ row, column ] == 0 );
106 } // end method ValidMove
107
108 // display Knight's tour path
109 public static void PrintTour()
110 {
111     Console.Write( "\n " );
112
113     // display numbers for column
114     for ( int k = 0; k < 8; k++ )
115         Console.Write( "{0,3}", k );
116
117     Console.WriteLine( "\n" );
118
119     for ( int row = 0; row < board.GetLength( 0 ); row++ )
120     {
121         Console.Write( "{0,-2}", row );
122
123         for ( int column = 0; column < board.GetLength( 1 ); column++ )
124             Console.Write( "{0,3}", board[ row, column ] );
125

```

```

126         Console.WriteLine();
127     } // end for
128 } // end method PrintTour
129 } // end class Knight2

```

The tour ended with 64 moves.
This was a full tour!

	0	1	2	3	4	5	6	7
0	6	35	4	43	8	33	28	55
1	3	44	7	34	29	54	9	32
2	36	5	42	49	46	31	56	27
3	41	2	45	30	57	48	53	10
4	20	37	50	47	52	59	26	61
5	1	40	19	58	23	62	11	14
6	18	21	38	51	16	13	60	25
7	39	1	17	22	63	24	15	12

- d) Write a version of the Knight's Tour application that, when encountering a tie between two or more squares, decides what square to choose by looking ahead to those squares reachable from the "tied" squares. Your application should move to the tied square for which the next move would arrive at the square with the lowest accessibility number.

ANS:

```

1 // Exercise 8.22 Part D Solution: Knight3.cs
2 // Knight's Tour - heuristic version
3 using System;
4
5 public class Knight3
6 {
7     static Random randomNumbers = new Random();
8
9     static int[ , ] access = { { 2, 3, 4, 4, 4, 4, 3, 2 },
10                                { 3, 4, 6, 6, 6, 6, 4, 3 },
11                                { 4, 6, 8, 8, 8, 8, 6, 4 },
12                                { 4, 6, 8, 8, 8, 8, 6, 4 },
13                                { 4, 6, 8, 8, 8, 8, 6, 4 },
14                                { 4, 6, 8, 8, 8, 8, 6, 4 },
15                                { 3, 4, 6, 6, 6, 6, 4, 3 },
16                                { 2, 3, 4, 4, 4, 4, 3, 2 } };
17
18     static int[ , ] board; // gameboard
19     static int accessNumber; // the current access number
20
21     // moves
22     static int[] horizontal = { 2, 1, -1, -2, -2, -1, 1, 2 };
23     static int[] vertical = { -1, -2, -2, -1, 1, 2, 2, 1 };
24
25     // create a board and attempt a tour
26     public static void Main( string[] args )
27     {

```

```

28     int currentRow; // the row position on the chessboard
29     int currentColumn; // the column position on the chessboard
30     int moveNumber = 0; // the current move number
31
32     int testRow; // row position of next possible move
33     int testColumn; // column position of next possible move
34     int minRow = -1; // row position of move with minimum access
35     int minColumn = -1; // row position of move with minimum access
36
37     board = new int[ 8, 8 ];
38
39     // randomize initial board position
40     currentRow = randomNumbers.Next( 8 );
41     currentColumn = randomNumbers.Next( 8 );
42
43     board[ currentRow, currentColumn ] = ++moveNumber;
44     bool done = false;
45
46     // continue touring until finished traversing
47     while ( !done )
48     {
49         accessNumber = 99;
50
51         // try all possible moves
52         for ( int moveType = 0; moveType < board.GetLength( 0 );
53             moveType++ )
54         {
55             // new position of hypothetical moves
56             testRow = currentRow + vertical[ moveType ];
57             testColumn = currentColumn + horizontal[ moveType ];
58
59             if ( ValidMove( testRow, testColumn ) )
60             {
61                 // obtain access number
62                 if ( access[ testRow, testColumn ] < accessNumber )
63                 {
64                     // if this is the lowest access number thus far,
65                     // then set this move to be our next move
66                     accessNumber = access[ testRow, testColumn ];
67
68                     minRow = testRow;
69                     minColumn = testColumn;
70                 } // end if
71                 else if
72                 ( access[ testRow, testColumn ] == accessNumber )
73                 {
74                     // if the lowest access numbers are the same,
75                     // look ahead to the next move to see which has the
76                     // lower access number
77                     int lowestTest = NextMove( testRow, testColumn );
78                     int lowestMin = NextMove( minRow, minColumn );
79
80                     if ( lowestTest <= lowestMin )
81                     {

```

```

82         accessNumber = access[ testRow, testColumn ];
83
84         minRow = testRow;
85         minColumn = testColumn;
86     } // end if
87 } // end else if
88
89     // position access number tried
90     --access[ testRow, testColumn ];
91 } // end if
92 } // end for
93
94 // traversing done
95 if ( accessNumber == 99 )
96     done = true;
97 else // make move
98 {
99     currentRow = minRow;
100    currentColumn = minColumn;
101    board[ currentRow, currentColumn ] = ++moveNumber;
102 } // end else
103 } // end while
104
105 Console.WriteLine( "The tour ended with {0} moves.", moveNumber );
106
107 if ( moveNumber == 64 )
108     Console.WriteLine( " This was a full tour!" );
109 else
110     Console.WriteLine( " This was not a full tour." );
111
112 PrintTour();
113 } // end method Tour
114
115 // checks for next move
116 public static int NextMove( int row, int column )
117 {
118     int tempRow, tempColumn;
119     int tempAccessNumber = accessNumber;
120     int[ , ] tempAccess = new int[ 8, 8 ];
121
122     for ( int i = 0; i < access.GetLength( 0 ); i++ )
123         for ( int j = 0; j < access.GetLength( 1 ); j++ )
124             tempAccess[ i, j ] = access[ i, j ];
125
126     // try all possible moves
127     for ( int moveType = 0; moveType < board.GetLength( 0 );
128         moveType++ )
129     {
130         // new position of hypothetical moves
131         tempRow = row + vertical[ moveType ];
132         tempColumn = column + horizontal[ moveType ];
133
134         if ( ValidMove( tempRow, tempColumn ) )
135         {

```

```
136         // obtain access number
137         if ( access[ tempRow, tempColumn ] < tempAccessNumber )
138             tempAccessNumber = tempAccess[ tempRow, tempColumn ];
139
140         // position access number tried
141         --tempAccess[ tempRow, tempColumn ];
142     } // end if
143 } // end for
144
145     return tempAccessNumber;
146 } // end method NextMove
147
148 // checks for valid move
149 public static bool ValidMove( int row, int column )
150 {
151     // returns false if the move is off the chessboard, or if
152     // the knight has already visited that position
153     // NOTE: This test stops as soon as it becomes false
154     return ( row >= 0 && row < 8 && column >= 0 && column < 8
155             && board[ row, column ] == 0 );
156 } // end method ValidMove
157
158 // display Knight's tour path
159 public static void PrintTour()
160 {
161     Console.Write( "\n " );
162
163     // display numbers for column
164     for ( int k = 0; k < 8; k++ )
165         Console.Write( "{0,3}", k );
166
167     Console.WriteLine( "\n" );
168
169     for ( int row = 0; row < board.GetLength( 0 ); row++ )
170     {
171         Console.Write( "{0,-2}", row );
172
173         for ( int column = 0; column < board.GetLength( 1 ); column++ )
174             Console.Write( "{0,3}", board[ row, column ] );
175
176         Console.WriteLine();
177     } // end for
178 } // end method PrintTour
179 } // end class Knight3
```

The tour ended with 64 moves.
This was a full tour!

	0	1	2	3	4	5	6	7
0	17	32	13	54	27	30	11	48
1	14	55	16	31	12	49	26	29
2	33	18	61	56	53	28	47	10
3	64	15	52	41	62	57	50	25
4	19	34	63	60	51	46	9	58
5	4	1	42	37	40	59	24	45
6	35	20	3	6	43	22	39	8
7	2	5	36	21	38	7	44	23

8.23 (*Knight's Tour: Brute-Force Approaches*) In part (c) of Exercise 8.22, we developed a solution to the Knight's Tour problem. The approach used, called the "accessibility heuristic," generates many solutions and executes efficiently.

As computers continue to increase in power, we'll be able to solve more problems with sheer computer power and relatively unsophisticated algorithms. Let us call this approach "brute-force" problem solving.

- Use random-number generation to enable the knight to walk around the chessboard (in its legitimate L-shaped moves) at random. Your application should run one tour and display the final chessboard. How far did the knight get?

ANS:

```

1 // Exercise 8.23 Part A Solution: Knight4.cs
2 // Knight's tour - Brute Force Approach. Uses random number
3 // generation to move around the board.
4 using System;
5
6 public class Knight4
7 {
8     static Random randomNumbers = new Random();
9
10    static int[ , ] board; // gameboard
11
12    // moves
13    static int[] horizontal = { 2, 1, -1, -2, -2, -1, 1, 2 };
14    static int[] vertical = { -1, -2, -2, -1, 1, 2, 2, 1 };
15
16    // runs a tour
17    public static void Main( string[] args )
18    {
19        int currentRow; // the row position on the chessboard
20        int currentColumn; // the column position on the chessboard
21        int moveNumber = 0; // the current move number
22
23        board = new int[ 8, 8 ]; // gameboard
24
25        int testRow; // row position of next possible move
26        int testColumn; // column position of next possible move
27
```

```
28 // randomize initial board position
29 currentRow = randomNumbers.Next( 8 );
30 currentColumn = randomNumbers.Next( 8 );
31
32 board[ currentRow, currentColumn ] = ++moveNumber;
33 bool done = false;
34
35 // continue until knight can no longer move
36 while ( !done )
37 {
38     bool goodMove = false;
39
40     // start with a random move
41     int moveType = randomNumbers.Next( 8 );
42
43     // check all possible moves until we find one that's legal
44     for ( int count = 0; count < 8 && !goodMove; count++ )
45     {
46         testRow = currentRow + vertical[ moveType ];
47         testColumn = currentColumn + horizontal[ moveType ];
48         goodMove = ValidMove( testRow, testColumn );
49
50         // test if new move is valid
51         if ( goodMove )
52         {
53             currentRow = testRow;
54             currentColumn = testColumn;
55             board[ currentRow, currentColumn ] = ++moveNumber;
56         } // end if
57
58         moveType = ( moveType + 1 ) % 8;
59     } // end for
60
61     // if no valid moves, knight can no longer move
62     if ( !goodMove )
63         done = true;
64     // if 64 moves have been made, a full tour is complete
65     else if ( moveNumber == 64 )
66         done = true;
67 } // end while
68
69 Console.WriteLine( "The tour ended with {0} moves.", moveNumber );
70
71 if ( moveNumber == 64 )
72     Console.WriteLine( "This was a full tour!" );
73 else
74     Console.WriteLine( "This was not a full tour." );
75
76 PrintTour();
77 } // end Main
78
79 // checks for valid move
80 public static bool ValidMove( int row, int column )
81 {
```

```

82      // returns false if the move is off the chessboard, or if
83      // the knight has already visited that position
84      // NOTE: This test stops as soon as it becomes false
85      return ( row >= 0 && row < 8 && column >= 0 && column < 8
86              && board[ row, column ] == 0 );
87  } // end method ValidMove
88
89  // display Knight's tour path
90  public static void PrintTour()
91  {
92      Console.Write( "\n " );
93
94      // display numbers for column
95      for ( int k = 0; k < 8; k++ )
96          Console.Write( "{0,3}", k );
97
98      Console.WriteLine( "\n" );
99
100     for ( int row = 0; row < board.GetLength( 0 ); row++ )
101     {
102         Console.Write( "{0,-2}", row );
103
104         for ( int column = 0; column < board.GetLength( 1 ); column++ )
105             Console.Write( "{0,3}", board[ row, column ] );
106
107         Console.WriteLine();
108     } // end for
109 } // end method PrintTour
110 } // end class Knight4

```

The tour ended with 34 moves.
This was not a full tour.

	0	1	2	3	4	5	6	7
0	0	0	0	5	18	21	24	3
1	0	6	19	0	25	4	17	22
2	0	0	0	0	20	23	2	0
3	7	0	9	12	0	26	0	16
4	0	13	28	0	10	15	0	1
5	31	8	11	14	27	0	0	0
6	0	0	32	29	0	0	34	0
7	0	30	0	0	33	0	0	0

- b) Most likely, the application in part (a) produced a relatively short tour. Now modify your application to attempt 1000 tours. Use a one-dimensional array to keep track of the number of tours of each length. When your application finishes attempting the 1000 tours, it should display this information in neat tabular format. What was the best result?

ANS:

```

1 // Exercise 8.23 Part B Solution: Knight5.cs
2 // Knight's tour - Brute Force Approach. Use random
3 // number generation to traverse the board. ( 1000 tours )
4 using System;
5
6 public class Knight5
7 {
8     static Random randomNumbers = new Random();
9
10    static int[ , ] board = new int[ 8, 8 ]; // gameboard
11
12    // moves
13    static int[] horizontal = { 2, 1, -1, -2, -2, -1, 1, 2 };
14    static int[] vertical = { -1, -2, -2, -1, 1, 2, 2, 1 };
15
16    static int[] moveTotals = new int[ 65 ]; // total # of tours per move
17
18    // runs a tour
19    public static void Main( string[] args )
20    {
21        int currentRow; // the row position on the chessboard
22        int currentColumn; // the column position on the chessboard
23
24        int testRow; // row position of next possible move
25        int testColumn; // column position of next possible move
26
27        for ( int k = 0; k < 1000; k++ )
28        {
29            ClearBoard();
30            int moveNumber = 0; // the current move number
31
32            // randomize initial board position
33            currentRow = randomNumbers.Next( 8 );
34            currentColumn = randomNumbers.Next( 8 );
35
36            board[ currentRow, currentColumn ] = ++moveNumber;
37            bool done = false;
38
39            // continue until knight can no longer move
40            while ( !done )
41            {
42                bool goodMove = false;
43
44                int moveType = randomNumbers.Next( 8 );
45
46                // check all possible moves until we find one that's legal
47                for ( int count = 0; count < 8 && !goodMove; count++ )
48                {
49                    testRow = currentRow + vertical[ moveType ];
50                    testColumn = currentColumn + horizontal[ moveType ];
51                    goodMove = ValidMove( testRow, testColumn );
52                }

```

```

53         // test if new move is valid
54         if ( goodMove )
55         {
56             currentRow = testRow;
57             currentColumn = testColumn;
58             board[ currentRow, currentColumn ] = ++moveNumber;
59         } // end if
60
61         moveType = ( moveType + 1 ) % 8;
62     } // end for
63
64     // if no valid moves, knight can no longer move
65     if ( !goodMove )
66         done = true;
67     // if 64 moves have been made, a full tour is complete
68     else if ( moveNumber == 64 )
69         done = true;
70 } // end while
71
72     ++moveTotals[ moveNumber ]; // update the statistics
73 } // end for
74
75     PrintResults();
76 } // end method Tour
77
78 // checks for valid move
79 public static bool ValidMove( int row, int column )
80 {
81     // returns false if the move is off the chessboard, or if
82     // the knight has already visited that position
83     // NOTE: This test stops as soon as it becomes false
84     return ( row >= 0 && row < 8 && column >= 0 && column < 8
85             && board[ row, column ] == 0 );
86 } // end method ValidMove
87
88 // display results
89 public static void PrintResults()
90 {
91     Console.Write( "# tours having # moves " );
92     Console.WriteLine( "# tours having # moves\n" );
93
94     // display results in tabulated columns
95     for ( int row = 1; row < 33; row++ )
96     {
97         Console.WriteLine( "{0,-15}{1,-9}{2,-15}{3}", moveTotals[ row ],
98             row, moveTotals[ row + 32 ], ( row + 32 ) );
99     } // end for
100 } // end method PrintResults
101
102 // resets board
103 public static void ClearBoard()
104 {

```

```

105     for ( int j = 0; j < board.GetLength( 0 ); j++ )
106         for ( int k = 0; k < board.GetLength( 1 ); k++ )
107             board[ j, k ] = 0;
108     } // end method ClearBoard
109 } // end class Knight5

```

# tours having	# moves	# tours having	# moves
0	1	20	33
0	2	37	34
0	3	29	35
1	4	36	36
0	5	41	37
0	6	30	38
1	7	42	39
3	8	27	40
1	9	26	41
4	10	38	42
1	11	38	43
6	12	27	44
7	13	22	45
7	14	43	46
6	15	35	47
12	16	36	48
5	17	24	49
14	18	23	50
4	19	20	51
12	20	27	52
10	21	12	53
16	22	17	54
14	23	7	55
17	24	13	56
12	25	2	57
23	26	4	58
21	27	1	59
29	28	0	60
24	29	2	61
21	30	1	62
17	31	0	63
32	32	0	64

- c) Most likely, the application in part (b) gave you some “respectable” tours, but no full tours. Now let your application run until it produces a full tour. Once again, keep a table of the number of tours of each length, and display this table when the first full tour is found. How many tours did your application attempt before producing a full tour?

ANS:

```

1 // Exercise 8.23 Part C Solution: Knight6.cs
2 // Knight's tour - Brute Force Approach. Use random
3 // number generation to traverse the board until a full tour is found
4 using System;
5
6 public class Knight6
7 {

```

```

8      static Random randomNumbers = new Random();
9
10     static int[ , ] board = new int[ 8, 8 ]; // gameboard
11
12     // moves
13     static int[] horizontal = { 2, 1, -1, -2, -2, -1, 1, 2 };
14     static int[] vertical = { -1, -2, -2, -1, 1, 2, 2, 1 };
15
16     static int[] moveTotals = new int[ 65 ]; // total # of tours per move
17
18     // runs a tour
19     public static void Main( string[] args )
20     {
21         int currentRow; // the row position on the chessboard
22         int currentColumn; // the column position on the chessboard
23
24         int testRow; // row position of next possible move
25         int testColumn; // column position of next possible move
26
27         bool fullTour = false;
28
29         while ( !fullTour )
30         {
31             ClearBoard();
32             int moveNumber = 0; // the current move number
33
34             // randomize initial board position
35             currentRow = randomNumbers.Next( 8 );
36             currentColumn = randomNumbers.Next( 8 );
37
38             board[ currentRow, currentColumn ] = ++moveNumber;
39             bool done = false;
40
41             // continue until knight can no longer move
42             while ( !done )
43             {
44                 bool goodMove = false;
45
46                 int moveType = randomNumbers.Next( 8 );
47
48                 // check all possible moves until we find one that's legal
49                 for ( int count = 0; count < 8 && !goodMove; count++ )
50                 {
51                     testRow = currentRow + vertical[ moveType ];
52                     testColumn = currentColumn + horizontal[ moveType ];
53                     goodMove = ValidMove( testRow, testColumn );
54
55                     // test if new move is valid
56                     if ( goodMove )
57                     {
58                         currentRow = testRow;
59                         currentColumn = testColumn;
60                         board[ currentRow, currentColumn ] = ++moveNumber;
61                     } // end if

```

```

62
63         moveType = ( moveType + 1 ) % 8;
64     } // end for
65
66     // if no valid moves, knight can no longer move
67     if ( !goodMove )
68         done = true;
69     // if 64 moves have been made, a full tour is complete
70     else if ( moveNumber == 64 )
71     {
72         done = true;
73         fullTour = true;
74     } // end else if
75 } // end while
76
77 ++moveTotals[ moveNumber ]; // update the statistics
78
79 } // end for
80
81 PrintResults();
82 } // end method Tour
83
84 // checks for valid move
85 public static bool ValidMove( int row, int column )
86 {
87     // returns false if the move is off the chessboard, or if
88     // the knight has already visited that position
89     // NOTE: This test stops as soon as it becomes false
90     return ( row >= 0 && row < 8 && column >= 0 && column < 8
91         && board[ row, column ] == 0 );
92 } // end method ValidMove
93
94 // display results
95 public static void PrintResults()
96 {
97     int totalTours = 0; // total number of moves
98
99     Console.Write( "# tours having # moves " );
100    Console.WriteLine( "# tours having # moves\n" );
101
102    // display results in tabulated columns
103    for ( int row = 1; row < 33; row++ )
104    {
105        Console.WriteLine( "{0,-15}{1,-9}{2,-15}{3}", moveTotals[ row ],
106            row, moveTotals[ row + 32 ], ( row + 32 ) );
107
108        totalTours += moveTotals[ row ] + moveTotals[ row + 32 ];
109    } // end for
110
111    Console.WriteLine( "\nIt took {0} tries to get a full tour",
112        totalTours );
113 } // end method PrintResults
114

```

```

115 // resets board
116 public static void ClearBoard()
117 {
118     for ( int j = 0; j < board.GetLength( 0 ); j++ )
119         for ( int k = 0; k < board.GetLength( 1 ); k++ )
120             board[ j, k ] = 0;
121 } // end method ClearBoard
122 } // end class Knight6

```

# tours having	# moves	# tours having	# moves
0	1	636	33
0	2	783	34
0	3	676	35
15	4	802	36
16	5	771	37
48	6	941	38
36	7	802	39
77	8	970	40
64	9	805	41
83	10	922	42
99	11	784	43
135	12	907	44
115	13	776	45
162	14	834	46
154	15	687	47
202	16	794	48
184	17	637	49
253	18	646	50
222	19	493	51
318	20	475	52
311	21	362	53
394	22	324	54
345	23	241	55
416	24	192	56
384	25	144	57
518	26	113	58
463	27	54	59
553	28	53	60
538	29	22	61
601	30	8	62
566	31	0	63
734	32	1	64

It took 24661 tries to get a full tour

- d) Compare the brute-force version of the Knight's Tour with the accessibility-heuristic version. Which required a more careful study of the problem? Which algorithm was more difficult to develop? Which required more computer power? Could we be certain (in advance) of obtaining a full tour with the accessibility-heuristic approach? Could we be certain (in advance) of obtaining a full tour with the brute-force approach? Argue the pros and cons of brute-force problem solving in general.

8.24 (*Eight Queens*) Another puzzler for chess buffs is the Eight Queens problem, which asks the following: Is it possible to place eight queens on an empty chessboard so that no queen is "attacking"

```

27     int currentRow; // the row position on the chessboard
28     int currentColumn; // the column position on the chessboard
29
30     board = new bool[ 8, 8 ]; // all elements default to false
31
32     // randomize initial first queen position
33     currentRow = randomNumbers.Next( 8 );
34     currentColumn = randomNumbers.Next( 8 );
35
36     board[ currentRow, currentColumn ] = true;
37     ++queens;
38
39     UpdateAccess( currentRow, currentColumn ); // update access
40
41     bool done = false;
42
43     // continue until finished traversing
44     while ( !done )
45     {
46         // the current lowest access number
47         int accessNumber = maxAccess;
48
49         // find square with the smallest elimination number
50         for ( int row = 0; row < board.GetLength( 0 ); row++ )
51         {
52             for ( int col = 0; col < board.GetLength( 1 ); col++ )
53             {
54                 // obtain access number
55                 if ( access[ row, col ] < accessNumber )
56                 {
57                     accessNumber = access[ row, col ];
58                     currentRow = row;
59                     currentColumn = col;
60                 } // end if
61             } // end inner for
62         } // end outer for
63
64         // traversing done
65         if ( accessNumber == maxAccess )
66             done = true;
67         // mark the current location
68         else
69         {
70             board[ currentRow, currentColumn ] = true;
71             UpdateAccess( currentRow, currentColumn );
72             ++queens;
73         } // end else
74     } // end while
75
76     PrintBoard();
77 } // end Main
78
79 // update access array
80 public static void UpdateAccess( int row, int column )
81 {

```

```

82     for ( int i = 0; i < 8; i++ )
83     {
84         // set elimination numbers to 99
85         // in the row occupied by the queen
86         access[ row, i ] = maxAccess;
87
88         // set elimination numbers to 99
89         // in the column occupied by the queen
90         access[ i, column ] = maxAccess;
91     } // end for
92
93     // set elimination numbers to 99 in diagonals occupied by the queen
94     UpdateDiagonals( row, column );
95 } // end method UpdateAccess
96
97 // place 99 in diagonals of position in all 4 directions
98 public static void UpdateDiagonals( int rowValue, int colValue )
99 {
100     int row = rowValue; // row position to be updated
101     int column = colValue; // column position to be updated
102
103     // upper left diagonal
104     for ( int diagonal = 0; diagonal < 8 &&
105         ValidMove( --row, --column ); diagonal++ )
106         access[ row, column ] = maxAccess;
107
108     row = rowValue;
109     column = colValue;
110
111     // upper right diagonal
112     for ( int diagonal = 0; diagonal < 8 &&
113         ValidMove( --row, ++column ); diagonal++ )
114         access[ row, column ] = maxAccess;
115
116     row = rowValue;
117     column = colValue;
118
119     // lower left diagonal
120     for ( int diagonal = 0; diagonal < 8 &&
121         ValidMove( ++row, --column ); diagonal++ )
122         access[ row, column ] = maxAccess;
123
124     row = rowValue;
125     column = colValue;
126
127     // lower right diagonal
128     for ( int diagonal = 0; diagonal < 8 &&
129         ValidMove( ++row, ++column ); diagonal++ )
130         access[ row, column ] = maxAccess;
131 } // end method UpdateDiagonals
132
133 // check for valid move
134 public static bool ValidMove( int row, int column )
135 {

```

```

136     return ( row >= 0 && row < 8 && column >= 0 && column < 8 );
137 } // end method ValidMove
138
139 // display the board
140 public static void PrintBoard()
141 {
142     Console.Write( " " );
143
144     // display numbers for column
145     for ( int k = 0; k < 8; k++ )
146         Console.Write( " {0}", k );
147
148     Console.WriteLine( "\n" );
149
150     for ( int row = 0; row < board.GetLength( 0 ); row++ )
151     {
152         Console.Write( "{0} ", row );
153
154         for ( int column = 0; column < board.GetLength( 1 ); column++ )
155         {
156             if ( board[ row, column ] )
157                 Console.Write( " Q" );
158             else
159                 Console.Write( " ." );
160         } // end for
161
162         Console.WriteLine();
163     } // end for
164
165     Console.WriteLine( "\n{0} queens placed on the board.", queens );
166 } // end method PrintBoard
167 } // end class EightQueens

```

```

    0 1 2 3 4 5 6 7
0  Q . . . . . .
1  . . . . . Q .
2  . . . . . . Q
3  . . Q . . . .
4  . . . . . Q .
5  . . . . . . .
6  . . . . . . .
7  . Q . . . . .

6 queens placed on the board.

```

8.25 (*Eight Queens: Brute-Force Approaches*) In this exercise, you'll develop several brute-force approaches to solving the Eight Queens problem introduced in Exercise 8.24.

- a) Use the random brute-force technique developed in Exercise 8.23 to solve the Eight Queens problem.

ANS:

```

1 // Exercise 8.25 PartA Solution: EightQueens1.cs
2 // Uses a random brute force approach to solve the eight queens problem.
3 using System;
4
5 public class EightQueens1
6 {
7     static Random randomNumbers = new Random();
8
9     static char[ , ] board; // chess board
10    static int queens; // number of queens placed
11
12    // place queens on board
13    public static void Main( string[] args )
14    {
15        // repeat until solved
16        while ( queens < 8 )
17        {
18            int rowMove; // column move
19            int colMove; // row move
20            bool done = false; // indicates if all squares filled
21
22            // reset the board
23            board = new char[ 8, 8 ];
24            queens = 0;
25
26            // continue placing queens until no more squares
27            // or not all queens placed
28            while ( !done )
29            {
30                // randomize move
31                rowMove = randomNumbers.Next( 8 );
32                colMove = randomNumbers.Next( 8 );
33
34                // if valid move, place queen and mark off conflict squares
35                if ( QueenCheck( rowMove, colMove ) )
36                {
37                    board[ rowMove, colMove ] = 'Q';
38                    XConflictSquares( rowMove, colMove );
39                    ++queens;
40                } // end if
41
42                // done when no more squares left
43                if ( !AvailableSquare() )
44                    done = true;
45            } // end inner while loop
46        } // end outer while loop
47
48        PrintBoard();
49    } // end method PlaceQueens
50
51    // check for valid move
52    public static bool ValidMove( int row, int column )
53    {

```

```

54     return ( row >= 0 && row < 8 && column >= 0 && column < 8 );
55 } // end method ValidMove
56
57 // check if any squares left
58 public static bool AvailableSquare()
59 {
60     for ( int row = 0; row < board.GetLength( 0 ); row++ )
61         for ( int col = 0; col < board.GetLength( 1 ); col++ )
62             if ( board[ row, col ] == '\0' )
63                 return true; // at least one available square
64
65     return false; // no available squares
66 } // end method AvailableSquare
67
68 // check if a queen can be placed without being attacked
69 public static bool QueenCheck( int rowValue, int colValue )
70 {
71     return ( board[ rowValue, colValue ] == '\0' );
72 } // end method QueenCheck
73
74 // conflicting square marked with *
75 public static void XConflictSquares( int row, int col )
76 {
77     for ( int i = 0; i < 8; i++ )
78     {
79         // place a '*' in the row occupied by the queen
80         if ( board[ row, i ] == '\0' )
81             board[ row, i ] = '*';
82
83         // place a '*' in the col occupied by the queen
84         if ( board[ i, col ] == '\0' )
85             board[ i, col ] = '*';
86     } // end for
87
88     // place a '*' in the diagonals occupied by the queen
89     XDiagonals( row, col );
90 } // end method XConflictSquares
91
92 // place * in diagonals of position in all 4 directions
93 public static void XDiagonals( int rowValue, int colValue )
94 {
95     int row = rowValue, column = colValue;
96
97     // upper left diagonal
98     for ( int diagonal = 0; diagonal < 8 &&
99         ValidMove( --row, --column ); diagonal++ )
100         board[ row, column ] = '*';
101
102     row = rowValue;
103     column = colValue;
104
105     // upper right diagonal
106     for ( int diagonal = 0; diagonal < 8 &&
107         ValidMove( --row, ++column ); diagonal++ )
108         board[ row, column ] = '*';

```

```

109
110     row = rowValue;
111     column = colValue;
112
113     // lower left diagonal
114     for ( int diagonal = 0; diagonal < 8 &&
115         ValidMove( ++row, --column ); diagonal++ )
116         board[ row, column ] = '*';
117
118     row = rowValue;
119     column = colValue;
120
121     // lower right diagonal
122     for ( int diagonal = 0; diagonal < 8 &&
123         ValidMove( ++row, ++column ); diagonal++ )
124         board[ row, column ] = '*';
125 } // end method XDiagonals
126
127 // displays the chessboard
128 public static void PrintBoard()
129 {
130     Console.Write( " " );
131
132     // display numbers for column
133     for ( int k = 0; k < 8; k++ )
134         Console.Write( " {0}", k );
135
136     Console.WriteLine( "\n" );
137
138     for ( int row = 0; row < board.GetLength( 0 ); row++ )
139     {
140         Console.Write( "{0} ", row );
141
142         for ( int column = 0; column < board.GetLength( 1 ); column++ )
143             Console.Write( " {0}", board[ row, column ] );
144
145         Console.WriteLine();
146     } // end for
147
148     Console.WriteLine( "\n{0} queens placed on the board.", queens );
149 } // end method PrintBoard
150 } // end class EightQueens1

```

```

0 1 2 3 4 5 6 7
0 * * Q * * * *
1 * * * * * Q *
2 * * * * * * Q
3 * Q * * * * *
4 * * * Q * * *
5 Q * * * * *
6 * * * * * Q *
7 * * * * Q * *

```

8 queens placed on the board.

- b) Use an exhaustive technique (i.e., try all possible combinations of eight queens on the chessboard) to solve the Eight Queens problem.

ANS:

```

1 // Exercise 8.25 Part B Solution: EightQueens2.cs
2 // Uses an exhaustive technique to solve the eight queens problem
3 using System;
4
5 public class EightQueens2
6 {
7     static char[ , ] board = new char[ 8, 8 ]; // chess board
8     static int queens; // number of queens placed
9
10    // place queens on board
11    public static void Main( string[] args )
12    {
13        for ( int firstQueenRow = 0;
14              firstQueenRow < board.GetLength( 0 ) && queens < 8;
15              firstQueenRow++ )
16        {
17            for ( int firstQueenCol = 0;
18                  firstQueenCol < board.GetLength( 1 ) && queens < 8;
19                  firstQueenCol++ )
20            {
21                // reset the board
22                board = new char[ 8, 8 ];
23                queens = 0;
24
25                // place first queen at starting position
26                board[ firstQueenRow, firstQueenCol ] = 'Q';
27                XConflictSquares( firstQueenRow, firstQueenCol );
28                ++queens;
29
30                // remaining queens will be placed in board
31
32                bool done = false; // indicates if all squares filled
33

```

```

34         // try all possible locations on board
35         for ( int rowMove = 0;
36             rowMove < board.GetLength( 0 ) && !done; rowMove++ )
37         {
38             for ( int colMove = 0;
39                 colMove < board.GetLength( 1 ) && !done; colMove++ )
40             {
41                 // if valid move, place queen
42                 // and mark off conflict squares
43                 if ( QueenCheck( rowMove, colMove ) )
44                 {
45                     board[ rowMove, colMove ] = 'Q';
46                     XConflictSquares( rowMove, colMove );
47                     ++queens;
48                 } // end if
49
50                 // done when no more squares left
51                 if ( !AvailableSquare() )
52                     done = true;
53             } // end for colMove
54         } // end for rowMove
55     } // end for firstQueenCol
56 } // end for firstQueenRow
57
58     PrintBoard();
59 } // end method PlaceQueens
60
61 // check for valid move
62 public static bool ValidMove( int row, int column )
63 {
64     return ( row >= 0 && row < 8 && column >= 0 && column < 8 );
65 } // end method ValidMove
66
67 // check if any squares left
68 public static bool AvailableSquare()
69 {
70     for ( int row = 0; row < board.GetLength( 0 ); row++ )
71         for ( int col = 0; col < board.GetLength( 1 ); col++ )
72             if ( board[ row, col ] == '\0' )
73                 return true; // at least one available square
74
75     return false; // no available squares
76 } // end method AvailableSquare
77
78 // conflicting square marked with *
79 public static void XConflictSquares( int row, int col )
80 {
81     for ( int i = 0; i < 8; i++ )
82     {
83
84         // place a '*' in the row occupied by the queen
85         if ( board[ row, i ] == '\0' )
86             board[ row, i ] = '*';
87

```

```

88         // place a '*' in the col occupied by the queen
89         if ( board[ i, col ] == '\0' )
90             board[ i, col ] = '*';
91     } // end for
92
93     // place a '*' in the diagonals occupied by the queen
94     XDiagonals( row, col );
95 } // end method XConflictSquares
96
97 // check if queens can "attack" each other
98 public static bool QueenCheck( int rowValue, int colValue )
99 {
100     return ( board[ rowValue, colValue ] == '\0' );
101 } // end method QueenCheck
102
103 // place * in diagonals of position in all 4 directions
104 public static void XDiagonals( int rowValue, int colValue )
105 {
106     int row = rowValue;
107     int column = colValue;
108
109     // upper left diagonal
110     for ( int diagonal = 0; diagonal < 8 &&
111         ValidMove( --row, --column ); diagonal++ )
112         board[ row, column ] = '*';
113
114     row = rowValue;
115     column = colValue;
116
117     // upper right diagonal
118     for ( int diagonal = 0; diagonal < 8 &&
119         ValidMove( --row, ++column ); diagonal++ )
120         board[ row, column ] = '*';
121
122     row = rowValue;
123     column = colValue;
124
125     // lower left diagonal
126     for ( int diagonal = 0; diagonal < 8 &&
127         ValidMove( ++row, --column ); diagonal++ )
128         board[ row, column ] = '*';
129
130     row = rowValue;
131     column = colValue;
132
133     // lower right diagonal
134     for ( int diagonal = 0; diagonal < 8 &&
135         ValidMove( ++row, ++column ); diagonal++ )
136         board[ row, column ] = '*';
137 } // end method XDiagonals
138
139 // displays the chessboard
140 public static void PrintBoard()
141 {

```

```

11             { 4, 6, 8, 8, 8, 8, 6, 4 },
12             { 4, 6, 8, 8, 8, 8, 6, 4 },
13             { 4, 6, 8, 8, 8, 8, 6, 4 },
14             { 4, 6, 8, 8, 8, 8, 6, 4 },
15             { 3, 4, 6, 6, 6, 6, 4, 3 },
16             { 2, 3, 4, 4, 4, 4, 3, 2 } };
17
18 static int[ , ] board; // gameboard
19 static int currentRow; // the row position on the chessboard
20 static int currentColumn; // the column position on the chessboard
21 static int firstRow; // the initial row position
22 static int firstColumn; // the initial column position
23 static int moveNumber = 0; // the current move number
24 static int accessNumber; // the current access number
25
26 // moves
27 static int[] horizontal = { 2, 1, -1, -2, -2, -1, 1, 2 };
28 static int[] vertical = { -1, -2, -2, -1, 1, 2, 2, 1 };
29
30 // create a board and attempt a tour
31 public static void Main( string[] args )
32 {
33     int testRow; // row position of next possible move
34     int testColumn; // column position of next possible move
35     int minRow = -1; // row position of move with minimum access
36     int minColumn = -1; // row position of move with minimum access
37
38     board = new int[ 8, 8 ];
39
40     // randomize initial board position
41     currentRow = randomNumbers.Next( 8 );
42     currentColumn = randomNumbers.Next( 8 );
43
44     firstRow = currentRow;
45     firstColumn = currentColumn;
46
47     board[ currentRow, currentColumn ] = ++moveNumber;
48     bool done = false;
49
50     // continue touring until finished traversing
51     while ( !done )
52     {
53         accessNumber = 99;
54
55         // try all possible moves
56         for ( int moveType = 0; moveType < board.GetLength( 0 );
57             moveType++ )
58         {
59             // new position of hypothetical moves
60             testRow = currentRow + vertical[ moveType ];
61             testColumn = currentColumn + horizontal[ moveType ];
62
63             if ( ValidMove( testRow, testColumn ) )
64             {

```

```

65         // obtain access number
66         if ( access[ testRow, testColumn ] < accessNumber )
67         {
68             // if this is the lowest access number thus far,
69             // then set this move to be our next move
70             accessNumber = access[ testRow, testColumn ];
71
72             minRow = testRow;
73             minColumn = testColumn;
74         } // end if
75
76         // position access number tried
77         --access[ testRow, testColumn ];
78     } // end if
79 } // end for
80
81 // traversing done
82 if ( accessNumber == 99 ) // no valid moves
83     done = true;
84 else // make move
85 {
86     currentRow = minRow;
87     currentColumn = minColumn;
88     board[ currentRow, currentColumn ] = ++moveNumber;
89 } // end else
90 } // end while
91
92 Console.WriteLine( "The tour ended with {0} moves.", moveNumber );
93
94 if ( moveNumber == 64 )
95 {
96     if ( ClosedTour() )
97         Console.WriteLine( " This was a CLOSED tour!" );
98     else
99         Console.WriteLine(
100             " This was a full tour, but it wasn't closed." );
101 } // end if
102 else
103     Console.WriteLine( " This was not a full tour." );
104
105 PrintTour();
106 } // end Main
107
108 // check for a closed tour if the last move can reach the initial
109 // starting position
110 public static bool ClosedTour()
111 {
112     // test all 8 possible moves to check if move
113     // would position knight on first move
114     for (int moveType = 0; moveType < board.GetLength( 0 ); moveType++)
115     {
116         int testRow = currentRow + vertical[ moveType ];
117         int testColumn = currentColumn + horizontal[ moveType ];
118

```

```

119         // if one move away from initial move
120         if ( testRow == firstRow &&
121             testColumn == firstColumn )
122         {
123             return true;
124         } // end if
125     } // end for
126
127     return false;
128 } // end method ClosedTour
129
130 // checks for valid move
131 public static bool ValidMove( int row, int column )
132 {
133     // returns false if the move is off the chessboard, or if
134     // the knight has already visited that position
135     // NOTE: This test stops as soon as it becomes false
136     return ( row >= 0 && row < 8 && column >= 0 && column < 8
137             && board[ row, column ] == 0 );
138 } // end method ValidMove
139
140 // display Knight's tour path
141 public static void PrintTour()
142 {
143     Console.WriteLine( "\n " );
144
145     // display numbers for column
146     for ( int k = 0; k < 8; k++ )
147         Console.WriteLine( "{0,3}", k );
148
149     Console.WriteLine( "\n" );
150
151     for ( int row = 0; row < board.GetLength( 0 ); row++ )
152     {
153         Console.WriteLine( "{0,-2}", row );
154
155         for ( int column = 0; column < board.GetLength( 1 ); column++ )
156             Console.WriteLine( "{0,3}", board[ row, column ] );
157
158         Console.WriteLine();
159     } // end for
160 } // end method PrintTour
161 } // end class Knight7

```

The tour ended with 64 moves.
This was a full tour, but it wasn't closed.

	0	1	2	3	4	5	6	7
0	11	8	13	40	49	6	57	52
1	14	39	10	7	58	51	48	5
2	9	12	41	50	43	64	53	56
3	38	15	44	31	54	59	4	47
4	33	30	37	42	63	46	55	22
5	16	27	32	45	36	21	60	3
6	29	34	25	18	1	62	23	20
7	26	17	28	35	24	19	2	61

8.27 (*Sieve of Eratosthenes*) A prime number is any integer greater than 1 that is evenly divisible only by itself and 1. The Sieve of Eratosthenes is a method of finding prime numbers. It operates as follows:

- Create a simple type `bool` array with all elements initialized to `true`. Array elements with prime indices will remain `true`. All other array elements will eventually be set to `false`.
- Starting with array index 2, determine whether a given element is `true`. If so, loop through the remainder of the array and set to `false` every element whose index is a multiple of the index for the element with value `true`. Then continue the process with the next element with value `true`. For array index 2, all elements beyond element 2 in the array with indices that are multiples of 2 (indices 4, 6, 8, 10, etc.) will be set to `false`; for array index 3, all elements beyond element 3 in the array with indices that are multiples of 3 (indices 6, 9, 12, 15, etc.) will be set to `false`; and so on.

When this process completes, the array elements that are still `true` indicate that the index is a prime number. These indices can be displayed. Write an application that uses an array of 1000 elements to determine and display the prime numbers between 2 and 999. Ignore array elements 0 and 1.

ANS:

```

1 // Exercise 8.27 Solution: Sieve.cs
2 // Sieve of Eratosthenes
3 using System;
4
5 public class Sieve
6 {
7     public static void Main( string[] args )
8     {
9         int count = 0; // the number of primes found
10
11         bool[] primes = new bool[ 1000 ]; // array of primes
12
13         // initialize all array values to true
14         for ( int index = 0; index < primes.Length; index++ )
15             primes[ index ] = true;
16

```

```

17      // starting at the third value, cycle through the array and put
18      // false as the value of any greater number that is a multiple
19      for ( int i = 2; i < primes.Length; i++ )
20          if ( primes[ i ] )
21              {
22                  for ( int j = i + i; j < primes.Length; j += i )
23                      primes[ j ] = false;
24              } // end if
25
26      // cycle through the array one last time to display all primes
27      for ( int index = 2; index < primes.Length; index++ )
28          if ( primes[ index ] )
29              {
30                  Console.WriteLine( "{0} is prime.", index );
31                  ++count;
32              } // end if
33
34      Console.WriteLine( "\n{0} primes found.", count );
35  } // end Main
36 } // end class Sieve

```

```

2 is prime.
3 is prime.
5 is prime.
7 is prime.

.
.
.

977 is prime.
983 is prime.
991 is prime.
997 is prime.

168 primes found.

```

8.28 (*Simulation: The Tortoise and the Hare*) In this problem, you'll re-create the classic race of the tortoise and the hare. You'll use random-number generation to develop a simulation of this memorable event.

Our contenders begin the race at square 1 of 70 squares. Each square represents a possible position along the race course. The finish line is at square 70. The first contender to reach or pass square 70 is rewarded with a pail of fresh carrots and lettuce. The course weaves its way up the side of a slippery mountain, so occasionally the contenders lose ground.

A clock ticks once per second. With each tick of the clock, your application should adjust the position of the animals according to the rules in Fig. 8.33. Use variables to keep track of the positions of the animals (i.e., position numbers are 1–70). Start each animal at position 1 (the “starting gate”). If an animal slips left before square 1, move it back to square 1.

Animal	Move type	Percentage of the time	Actual move
Tortoise	Fast plod	50%	3 squares to the right
	Slip	20%	6 squares to the left
	Slow plod	30%	1 square to the right
Hare	Sleep	20%	No move at all
	Big hop	20%	9 squares to the right
	Big slip	10%	12 squares to the left
	Small hop	30%	1 square to the right
	Small slip	20%	2 squares to the left

Fig. 8.28 | Rules for adjusting the positions of the tortoise and the hare.

Generate the percentages in Fig. 8.33 by producing a random integer i in the range $1 \leq i \leq 10$. For the tortoise, perform a “fast plod” when $1 \leq i \leq 5$, a “slip” when $6 \leq i \leq 7$ or a “slow plod” when $8 \leq i \leq 10$. Use a similar technique to move the hare.

Begin the race by displaying

```
ON YOUR MARK, GET SET
BANG !!!!!
AND THEY'RE OFF !!!!!
```

Then, for each tick of the clock (i.e., each repetition of a loop), display a 70-position line showing the letter T in the position of the tortoise and the letter H in the position of the hare. Occasionally, the contenders will land on the same square. In this case, the tortoise bites the hare, and your application should display OUCH!!! beginning at that position. All output positions other than the T, the H or the OUCH!!! (in case of a tie) should be blank.

After each line is displayed, test for whether either animal has reached or passed square 70. If so, display the winner and terminate the simulation. If the tortoise wins, display TORTOISE WINS!!! YAY!!! If the hare wins, display Hare wins. Yuch. If both animals win on the same tick of the clock, you may want to favor the tortoise (the “underdog”), or you may want to display It's a tie. If neither animal wins, perform the loop again to simulate the next tick of the clock. When you are ready to run your application, assemble a group of fans to watch the race. You'll be amazed at how involved your audience gets!

Later in the book, we introduce a number of C# capabilities, such as graphics, images, animation, sound and multithreading. As you study those features, you might enjoy enhancing your tortoise-and-hare contest simulation.

ANS:

```
1 // Exercise 8.28 Solution: Race.cs
2 // Application simulates the race between the tortoise and the hare
3 using System;
4
5 public class Race
6 {
7     const int RACE_END = 70; // const position
8
9     static Random randomNumbers = new Random();
10
```

```

11  static int tortoise; // tortoise's position
12  static int hare; // hare's position
13  static int timer; // clock ticks elapsed
14
15  // run the race
16  public static void Main( string[] args )
17  {
18      tortoise = 1;
19      hare = 1;
20      timer = 0;
21
22      Console.WriteLine( "ON YOUR MARK, GET SET" );
23      Console.WriteLine( "BANG !!!!!" );
24      Console.WriteLine( "AND THEY'RE OFF !!!!!" );
25
26      while ( tortoise < RACE_END && hare < RACE_END )
27      {
28          MoveHare();
29          MoveTortoise();
30          PrintCurrentPositions();
31
32          // slow down race
33          for ( int temp = 0; temp < 100000000; temp++ ) ;
34
35          ++timer;
36      } // end while
37
38      // tortoise beats hare or a tie
39      if ( tortoise >= hare )
40          Console.WriteLine( "\nTORTOISE WINS!!! YAY!!!" );
41      // hare beat tortoise
42      else
43          Console.WriteLine( "\nHare wins. Yuch!" );
44
45      Console.WriteLine( "TIME ELAPSED = {0} seconds", timer );
46  }
47
48  // move tortoise's position
49  public static void MoveTortoise()
50  {
51      // randomize move to choose
52      int percent = randomNumbers.Next( 1, 11 );
53
54      // determine moves by percent in range in Fig 8.33
55      // fast plod
56      if ( percent >= 1 && percent <= 5 )
57          tortoise += 3;
58      // slip
59      else if ( percent == 6 || percent == 7 )
60          tortoise -= 6;
61      // slow plod
62      else
63          ++tortoise;
64

```

```

65         // ensure tortoise doesn't slip beyond start position
66         if ( tortoise < 1 )
67             tortoise = 1;
68
69         // ensure tortoise doesn't pass the finish
70         else if ( tortoise > RACE_END )
71             tortoise = RACE_END;
72     } // end method MoveTortoise
73
74     // move hare's position
75     public static void MoveHare()
76     {
77         // randomize move to choose
78         int percent = randomNumbers.Next( 1, 11 );
79
80         // determine moves by percent in range in Fig 8.33
81         // big hop
82         if ( percent == 3 || percent == 4 )
83             hare += 9;
84         // big slip
85         else if ( percent == 5 )
86             hare -= 12;
87         // small hop
88         else if ( percent >= 6 && percent <= 8 )
89             ++hare;
90         // small slip
91         else if ( percent > 8 )
92             hare -= 2;
93
94         // ensure that hare doesn't slip beyond start position
95         if ( hare < 1 )
96             hare = 1;
97         // ensure hare doesn't pass the finish
98         else if ( hare > RACE_END )
99             hare = RACE_END;
100    } // end method MoveHare
101
102    // display positions of tortoise and hare
103    public static void PrintCurrentPositions()
104    {
105        // goes through all 70 squares, printing H
106        // if hare on position and T for tortoise
107        for ( int count = 1; count <= RACE_END; count++ )
108            // tortoise and hare positions collide
109            if ( count == tortoise && count == hare )
110                Console.Write( "OUCH!!!" );
111            else if ( count == hare )
112                Console.Write( "H" );
113            else if ( count == tortoise )
114                Console.Write( "T" );
115            else
116                Console.Write( " " );
117
118        Console.WriteLine();
119    } // end PrintCurrentPositions

```

```
120 } // end class Race
```

ON YOUR MARK, GET SET
BANG !!!!!
AND THEY'RE OFF !!!!!

[illegible]

OUCH!!!

Hare wins. Yuch!
TIME ELAPSED = 45 seconds

8.29 (*Card Shuffling and Dealing*) Modify the application of Fig. 8.11 to deal a five-card poker hand. Then modify class `DeckOfCards` of Fig. 8.10 to include methods that determine whether a hand contains

- a) a pair
- b) two pairs
- c) three of a kind (e.g., three jacks)
- d) four of a kind (e.g., four aces)
- e) a flush (i.e., all five cards of the same suit)
- f) a straight (i.e., five cards of consecutive face values)
- g) a full house (i.e., two cards of one face value and three cards of another face value)

[Hint: Add methods GetFace and GetSuit to class Card of Fig. 8.9.]

ANS:

```

1 // Exercise 8.29 Solution: Card.cs
2 // Card class represents a playing card.
3
4 public class Card
5 {
6     private string face; // face of card
7     private string suit; // suit of card
8
9     // two-parameter constructor initializes card's face and suit
10    public Card( string cardFace, string cardSuit )
11    {
12        face = cardFace; // initialize face of card
13        suit = cardSuit; // initialize suit of card
14    } // end two-parameter Card constructor
15
16    // return card face
17    public string GetFace()
18    {
19        return face;
20    } // end method GetFace
21
22    // return card suit
23    public string GetSuit()
24    {
25        return suit;
26    } // end method GetSuit
27
28    // return string representation of Card
29    public override string ToString()
30    {
31        return face + " of " + suit;
32    } // end method ToString
33 } // end class Card

```

```

1 // Exercise 8.29 Solution: DeckOfCards.cs
2 // DeckOfCards class represents a deck of playing cards.
3 using System;
4
5 public class DeckOfCards
6 {
7     string[] faces = { "Ace", "Deuce", "Three", "Four", "Five", "Six",
8         "Seven", "Eight", "Nine", "Ten", "Jack", "Queen", "King" };

```

```

 9  string[] suits = { "Hearts", "Diamonds", "Clubs", "Spades" };
10  private Card[] deck; // array of Card objects
11  private int currentCard; // the index of next Card to be dealt
12  private const int NUMBER_OF_CARDS = 52; // constant number of cards
13  private Random randomNumbers; // random number generator
14
15  // constructor fills deck of cards
16  public DeckOfCards()
17  {
18      deck = new Card[ NUMBER_OF_CARDS ]; // create array of Card objects
19      currentCard = 0; // initialize currentCard
20      randomNumbers = new Random(); // create random number generator
21
22      // populate deck with Card objects
23      for ( int count = 0; count < deck.Length; count++ )
24          deck[ count ] =
25              new Card( faces[ count % 13 ], suits[ count / 13 ] );
26  } // end DeckOfCards constructor
27
28  // shuffle deck of cards with one-pass algorithm
29  public void Shuffle()
30  {
31      currentCard = 0; // reinitialize currentCard
32
33      // for each card, pick another random card and swap them
34      for ( int first = 0; first < deck.Length; first++ )
35      {
36          int second = randomNumbers.Next( NUMBER_OF_CARDS );
37          Card temp = deck[ first ];
38          deck[ first ] = deck[ second ];
39          deck[ second ] = temp;
40      } // end for
41  } // end method Shuffle
42
43  // deal one card
44  public Card DealCard()
45  {
46      // determine whether cards remain to be dealt
47      if ( currentCard < deck.Length )
48          return deck[ currentCard++ ]; // return current Card in array
49      else
50          return null; // indicate that all cards were dealt
51  } // end method DealCard
52
53  // tally the number of each face card in hand
54  private int[] TotalHand( Card[] hand )
55  {
56      int[] numbers = new int[ faces.Length ]; // store number of face
57
58      // compare each card in the hand to each element in the faces array
59      for ( int h = 0; h < hand.Length; h++ )
60      {
61          for ( int f = 0; f < 13; f++ )
62          {

```

```
63         if ( hand[ h ].GetFace() == faces[ f ] )
64             ++numbers[ f ];
65     } // end for
66 } // end for
67
68     return numbers;
69 } // end method TotalHand
70
71 // determine if hand contains pairs
72 public int Pairs( Card[] hand )
73 {
74     int couples = 0;
75     int[] numbers = TotalHand( hand );
76
77     // count pairs
78     for ( int k = 0; k < numbers.Length; k++ )
79     {
80         if ( numbers[ k ] == 2 )
81         {
82             Console.WriteLine( "Pair of {0}s", faces[ k ] );
83             ++couples;
84         } // end if
85     } // end for
86
87     return couples;
88 } // end method Pairs
89
90 // determine if hand contains a three of a kind
91 public int ThreeOfAKind( Card[] hand )
92 {
93     int triples = 0;
94     int[] numbers = TotalHand( hand );
95
96     // count three of a kind
97     for ( int k = 0; k < numbers.Length; k++ )
98     {
99         if ( numbers[ k ] == 3 )
100         {
101             Console.WriteLine( "Three {0}s", faces[ k ] );
102             ++triples;
103             break;
104         } // end if
105     } // end for
106
107     return triples;
108 } // end method ThreeOfAKind
109
110 // determine if hand contains a four of a kind
111 public void FourOfAKind( Card[] hand )
112 {
113     int[] numbers = TotalHand( hand );
114
115     for ( int k = 0; k < faces.Length; k++ )
116     {
```

```

117         if ( numbers[ k ] == 4 )
118             Console.WriteLine( "Four {0}s", faces[ k ] );
119     } // end for
120 } // end FourOfAKind
121
122 // determine if hand contains a flush
123 public void Flush( Card[] hand )
124 {
125     string theSuit = hand[ 0 ].GetSuit();
126
127     for ( int s = 1; s < hand.Length; s++ )
128     {
129         if ( hand[ s ].GetSuit() != theSuit )
130             return; // not a flush
131     } // end for
132
133     Console.WriteLine( "Flush in {0}", theSuit );
134 } // end method Flush
135
136 // determine if hand contains a straight
137 public void Straight( Card[] hand )
138 {
139     int[] locations = new int[ 5 ];
140     int z = 0;
141     int[] numbers = TotalHand( hand );
142
143     for ( int y = 0; y < numbers.Length; y++ )
144     {
145         if ( numbers[ y ] == 1 )
146             locations[ z++ ] = y;
147     } // end for
148
149     int faceValue = locations[ 0 ];
150
151     if ( faceValue == 0 ) // special case, faceValue is Ace
152     {
153         faceValue = 13;
154
155         for ( int m = locations.Length - 1; m >= 1; m-- )
156         {
157             if ( faceValue != locations[ m ] + 1 )
158                 return; // not a straight
159             else
160                 faceValue = locations[ m ];
161         } // end if
162     } // end if
163     else
164     {
165         for ( int m = 1; m < locations.Length; m++ )
166         {
167             if ( faceValue != locations[ m ] - 1 )
168                 return; // not a straight

```

```

169         else
170             faceValue = locations[ m ];
171         } // end if
172     } // end else
173
174     Console.WriteLine( "Straight" );
175 } // end method Straight
176
177 // determine if hand contains a full house
178 public void FullHouse( int couples, int triples )
179 {
180     if ( couples == 1 && triples == 1 )
181         Console.WriteLine( "\nFull House!" );
182 } // end method FullHouse
183
184 // determine if hand contains two pairs
185 public void TwoPairs( int couples )
186 {
187     if ( couples == 2 )
188         Console.WriteLine( "\nTwo Pair!" );
189     } // end method TwoPairs
190 } // end class DeckOfCards

```

```

1 // Exercise 8.29 Solution: DeckOfCardsTest.cs
2 // Card shuffling and dealing application.
3 using System;
4
5 public class DeckOfCardsTest
6 {
7     // execute application
8     public static void Main( string[] args )
9     {
10         DeckOfCards myDeckOfCards = new DeckOfCards();
11         myDeckOfCards.Shuffle(); // place Cards in random order
12
13         Card[] hand = new Card[ 5 ]; // store five cards
14
15         // get first five cards
16         for ( int i = 0; i < 5; i++ )
17         {
18             hand[ i ] = myDeckOfCards.DealCard(); // get next card
19             Console.WriteLine( hand[ i ] );
20         } // end for
21
22         // display result
23         Console.WriteLine( "\nHand contains:" );
24
25         int couples = myDeckOfCards.Pairs( hand ); // a pair
26         myDeckOfCards.TwoPairs( couples ); // two pairs
27         // three of a kind
28         int triples = myDeckOfCards.ThreeOfAKind( hand );
29         myDeckOfCards.FourOfAKind( hand ); // four of a kind
30         myDeckOfCards.Flush( hand ); // a flush

```

```

31         myDeckOfCards.Straight( hand ); // a straight
32         myDeckOfCards.FullHouse( couples, triples ); // a full house
33     } // end Main
34 } // end class DeckOfCardsTest

```

```

Ten of Spades
Queen of Spades
Ace of Hearts
Jack of Spades
Ace of Diamonds

```

```

Hand contains:
Pair of Aces

```

```

Five of Clubs
Five of Spades
Seven of Hearts
Queen of Diamonds
Queen of Hearts

```

```

Hand contains:
Pair of Fives
Pair of Queens

```

```

Two Pair!

```

8.30 (*Card Shuffling and Dealing*) Use the methods developed in Exercise 8.29 to write an application that deals two five-card poker hands, evaluates each hand and determines which is the better hand.

ANS:

```

1  // Exercise 8.30 Solution: Card.cs
2  // Card class represents a playing card.
3
4  public class Card
5  {
6      private string face; // face of card
7      private string suit; // suit of card
8
9      // two-argument constructor initializes card's face and suit
10     public Card( string cardFace, string cardSuit )
11     {
12         face = cardFace; // initialize face of card
13         suit = cardSuit; // initialize suit of card
14     } // end two-argument Card constructor
15
16     // return card face
17     public string GetFace()
18     {
19         return face;
20     } // end method GetFace

```

```

21
22     // return card suit
23     public string GetSuit()
24     {
25         return suit;
26     } // end method GetSuit
27
28     // return string representation of Card
29     public override string ToString()
30     {
31         return face + " of " + suit;
32     } // end method ToString
33 } // end class Card

```

```

1 // Exercise 8.30 Solution: DeckOfCards.cs
2 // DeckOfCards class represents a deck of playing cards.
3 using System;
4
5 public class DeckOfCards
6 {
7     string[] faces = { "Ace", "Deuce", "Three", "Four", "Five", "Six",
8         "Seven", "Eight", "Nine", "Ten", "Jack", "Queen", "King" };
9     string[] suits = { "Hearts", "Diamonds", "Clubs", "Spades" };
10    private Card[] deck; // array of Card objects
11    private int currentCard; // the index of the next Card to be dealt
12    private const int NUMBER_OF_CARDS = 52; // constant number of cards
13    private Random randomNumbers; // random number generator
14    private bool straightHand1, straightHand2, pair1, pair2;
15    private int hand1Value, hand2Value;
16    private const int ONEPAIR = 2;
17    private const int TWOPAIR = 4;
18    private const int THREEKIND = 6;
19    private const int STRAIGHT = 8;
20    private const int FULLHOUSE = 10;
21    private const int FLUSH = 12;
22    private const int FOURKIND = 14;
23    private const int STRAIGHTFLUSH = 16;
24
25    // constructor fills deck of cards
26    public DeckOfCards()
27    {
28        deck = new Card[ NUMBER_OF_CARDS ]; // create array of Card objects
29        currentCard = 0; // initialize currentCard
30        randomNumbers = new Random(); // create random number generator
31
32        // populate deck with Card objects
33        for ( int count = 0; count < deck.Length; count++ )
34            deck[ count ] =
35                new Card( faces[ count % 13 ], suits[ count / 13 ] );
36    } // end DeckOfCards constructor
37

```

```

38 // shuffle deck of cards with one-pass algorithm
39 public void Shuffle()
40 {
41     currentCard = 0; // reinitialize currentCard
42
43     // for each card, pick another random card and swap them
44     for ( int first = 0; first < deck.Length; first++ )
45     {
46         int second = randomNumbers.Next( NUMBER_OF_CARDS );
47         Card temp = deck[ first ];
48         deck[ first ] = deck[ second ];
49         deck[ second ] = temp;
50     } // end for
51 } // end method Shuffle
52
53 // deal one card
54 public Card DealCard()
55 {
56     // determine whether cards remain to be dealt
57     if ( currentCard < deck.Length )
58         return deck[ currentCard++ ]; // return current Card in array
59     else
60         return null; // indicate that all cards were dealt
61 } // end method DealCard
62
63 // tally the number of each face card in hand
64 private int[] TotalHand( Card[] hand )
65 {
66     int[] numbers = new int[ faces.Length ]; // store number of face
67
68     // compare each card in the hand to each element in the faces array
69     for ( int h = 0; h < hand.Length; h++ )
70     {
71         for ( int f = 0; f < 13; f++ )
72         {
73             if ( hand[ h ].GetFace() == faces[ f ] )
74                 ++numbers[ f ];
75         } // end for
76     } // end for
77
78     return numbers;
79 } // end method TotalHand
80
81 // determine if hand contains pairs
82 public void Pairs( Card[] hand1, Card[] hand2 )
83 {
84     int numberPairs1 = 0; // number of pairs in hand1
85     int numberPairs2 = 0; // number of pairs in hand2
86     int highest1 = 0; // highest number of pair in hand1
87     int highest2 = 0; // highest number of pair in hand2
88     int[] numbers1 = TotalHand( hand1 ); // tally the number of each
89     int[] numbers2 = TotalHand( hand2 ); // face in hand1 and hand2
90

```

```
91 // count number of pairs in hands
92 for ( int k = 0; k < faces.Length; k++ )
93 {
94     // pair found in hand 1
95     if ( numbers1[ k ] == 2 )
96     {
97         pair1 = true;
98
99         // store highest pair
100         if ( k == 0 )
101             highest1 = 13; // special value for ace
102
103         if ( k > highest1 )
104             highest1 = k;
105
106         ++numberPairs1;
107     } // end if
108
109     // pair found in hand 2
110     if ( numbers2[ k ] == 2 )
111     {
112         pair2 = true;
113
114         // store highest pair
115         if ( k == 0 )
116             highest2 = 13; // special value for ace
117
118         if ( k > highest2 )
119             highest2 = k;
120
121         ++numberPairs2;
122     } // end if
123 } // end for
124
125 // evaluate number of pairs in each hand
126 if ( numberPairs1 == 1 )
127     hand1Value = ONEPAIR;
128 else if ( numberPairs1 == 2 )
129     hand1Value = TWOPAIR;
130
131 if ( numberPairs2 == 1 )
132     hand2Value = ONEPAIR;
133 else if ( numberPairs2 == 2 )
134     hand2Value = TWOPAIR;
135
136 if ( highest1 > highest2 )
137     ++hand1Value;
138 else if ( highest2 > highest1 )
139     ++hand2Value;
140 } // end method Pairs
141
142 // determine if hand contains a three of a kind
143 public void ThreeOfAKind( Card[] hand1, Card[] hand2 )
144 {
```

```

145     int tripletValue1 = 0; // highest triplet value in hand1
146     int tripletValue2 = 0; // highest triplet value in hand2
147     bool flag1 = false;
148     bool flag2 = false;
149     int[] numbers1 = TotalHand( hand1 ); // tally the number of each
150     int[] numbers2 = TotalHand( hand2 ); // face in hand1 and hand2
151
152     // check for three of a kind
153     for ( int k = 0; k < faces.Length; k++ )
154     {
155         // three of a kind found in hand 1
156         if ( numbers1[ k ] == 3 )
157         {
158             hand1Value = THREEKIND;
159             flag1 = true;
160
161             // store value of triplet
162             if ( k == 0 )
163                 tripletValue1 = 13; // special value for ace
164
165             if ( k > tripletValue1 )
166                 tripletValue1 = k;
167
168             if ( pair1 == true )
169                 hand1Value = FULLHOUSE;
170         } // end if
171
172         // three of a kind found in hand 2
173         if ( numbers2[ k ] == 3 )
174         {
175             hand2Value = THREEKIND;
176             flag2 = true;
177
178             // store value of triplet
179             if ( k == 0 )
180                 tripletValue2 = 13; // special value for ace
181
182             if ( k > tripletValue2 )
183                 tripletValue2 = k;
184
185             if ( pair2 == true )
186                 hand2Value = FULLHOUSE;
187         } // end if
188     } // end for
189
190     // both hands have three of a kind,
191     // determine which triplet is higher in value
192     if ( flag1 == true && flag2 == true )
193     {
194         if ( tripletValue1 > tripletValue2 )
195             ++hand1Value;
196         else if ( tripletValue1 < tripletValue2 )
197             ++hand2Value;
198     } // end if
199 } // end method ThreeOfAKind

```

```
200
201 // determine if hand contains a four of a kind
202 public void FourOfAKind( Card[] hand1, Card[] hand2 )
203 {
204     int highest1 = 0;
205     int highest2 = 0;
206     bool flag1 = false;
207     bool flag2 = false;
208     int[] numbers1 = TotalHand( hand1 ); // tally the number of each
209     int[] numbers2 = TotalHand( hand2 ); // face in hand1 and hand2
210
211     // check for four of a kind
212     for ( int k = 0; k < faces.Length; k++ )
213     {
214         // hand 1
215         if ( numbers1[ k ] == 4 )
216         {
217             hand1Value = FOURKIND;
218             flag1 = true;
219
220             if ( k == 0 )
221                 highest1 = 13; // special value for ace
222
223             if ( k > highest1 )
224                 highest1 = k;
225         } // end if
226
227         // hand 2
228         if ( numbers2[ k ] == 4 )
229         {
230             hand2Value = FOURKIND;
231             flag2 = true;
232
233             if ( k == 0 )
234                 highest2 = 13; // special value for ace
235
236             if ( k > highest2 )
237                 highest2 = k;
238         } // end if
239     } // end for
240
241     // if both hands contain four of a kind, determine which is higher
242     if ( flag1 == true && flag2 == true )
243     {
244         if ( highest1 > highest2 )
245             ++hand1Value;
246         else if ( highest1 < highest2 )
247             ++hand2Value;
248     } // end if
249 } // end FourOfAKind
250
251 // determine if hand contains a flush
252 public void Flush( Card[] hand1, Card[] hand2 )
253 {
```

```

254     string hand1Suit = hand1[ 0 ].GetSuit();
255     string hand2Suit = hand2[ 0 ].GetSuit();
256     bool flag1 = true, flag2 = true;
257
258     // check hand1
259     for ( int s = 1; s < hand1.Length && flag1 == true; s++ )
260     {
261         if ( hand1[ s ].GetSuit() != hand1Suit )
262             flag1 = false; // not a flush
263     } // end for
264
265     // check hand2
266     for ( int s = 1; s < hand2.Length && flag2 == true; s++ )
267     {
268         if ( hand2[ s ].GetSuit() != hand2Suit )
269             flag2 = false; // not a flush
270     } // end for
271
272     // hand 1 is a flush
273     if ( flag1 == true )
274     {
275         hand1Value = FLUSH;
276
277         // straight flush
278         if ( straightHand1 == true )
279             hand1Value = STRAIGHTFLUSH;
280     } // end if
281
282     // hand 2 is a flush
283     if ( flag2 == true )
284     {
285         hand2Value = FLUSH;
286
287         // straight flush
288         if ( straightHand2 == true )
289             hand2Value = STRAIGHTFLUSH;
290     } // end if
291 } // end method Flush
292
293 // determine if hand contains a straight
294 public void Straight( Card[] hand1, Card[] hand2 )
295 {
296     int[] locations1 = new int[ 5 ];
297     int[] locations2 = new int[ 5 ];
298     int[] numbers1 = TotalHand( hand1 ); // tally the number of each
299     int[] numbers2 = TotalHand( hand2 ); // face in hand1 and hand2
300
301     // check each card in both hands
302     for ( int y = 0, z = 0; y < numbers1.Length; y++ )
303     {
304         if ( numbers1[ y ] == 1 )
305             locations1[ z++ ] = y;
306     } // end for
307

```

```
308     for ( int y = 0, z = 0; y < numbers2.Length; y++ )
309     {
310         if ( numbers1[ y ] == 1 )
311             locations1[ z++ ] = y;
312     } // end for
313
314     int faceValue = locations1[ 0 ];
315     bool flag1 = true, flag2 = true;
316
317     if ( faceValue == 0 ) // special case, faceValue is Ace
318     {
319         faceValue = 13;
320
321         for ( int m = locations1.Length - 1; m >= 1; m-- )
322         {
323             if ( faceValue != locations1[ m ] + 1 )
324                 break; // not a straight
325             else
326                 faceValue = locations1[ m ];
327         } // end if
328     } // end if
329     else
330     {
331         for ( int m = 1; m < locations1.Length; m++ )
332         {
333             if ( faceValue != locations1[ m ] - 1 )
334                 break; // not a straight
335             else
336                 faceValue = locations1[ m ];
337         } // end if
338     } // end else
339
340     faceValue = locations2[ 0 ];
341
342     if ( faceValue == 0 ) // special case, faceValue is Ace
343     {
344         faceValue = 13;
345
346         for ( int m = locations2.Length - 1; m >= 1; m-- )
347         {
348             if ( faceValue != locations2[ m ] + 1 )
349                 break; // not a straight
350             else
351                 faceValue = locations2[ m ];
352         } // end if
353     } // end if
354     else
355     {
356         for ( int m = 1; m < locations2.Length; m++ )
357         {
358             if ( faceValue != locations2[ m ] - 1 )
359                 break; // not a straight
```

```

360         else
361             faceValue = locations2[ m ];
362         } // end if
363     } // end else
364
365     int highest1 = 0;
366     int highest2 = 0;
367
368     // hand 1 is a straight
369     if ( flag1 == true )
370     {
371         straightHand1 = true;
372         hand1Value = STRAIGHT;
373
374         if ( locations1[ 0 ] != 0 )
375             highest1 = locations1[ 4 ];
376         else
377             highest1 = 13;
378     } // end if
379
380     // hand 2 is a straight
381     if ( flag2 == true )
382     {
383         straightHand2 = true;
384         hand2Value = STRAIGHT;
385
386         if ( locations2[ 0 ] != 0 )
387             highest2 = locations2[ 4 ];
388         else
389             highest2 = 13;
390     } // end if
391
392     // if both hands contain straights,
393     // determine which is higher
394     if ( straightHand1 == true && straightHand2 == true )
395     {
396         if ( highest1 > highest2 )
397             ++hand1Value;
398         else if ( highest2 > highest1 )
399             ++hand2Value;
400     } // end if
401 } // end method Straight
402
403 // compare two hands
404 public void CompareTwoHands( Card[] hand1, Card[] hand2 )
405 {
406     // calculate contents of the two hand
407     Pairs( hand1, hand2 );
408     ThreeOfAKind( hand1, hand2 );
409     FourOfAKind( hand1, hand2 );
410     Straight( hand1, hand2 );
411     Flush( hand1, hand2 );
412     DisplayHandValues(); // display hand values
413

```

```
414     int[] numbers1 = TotalHand( hand1 ); // tally the number of each
415     int[] numbers2 = TotalHand( hand2 ); // face in hand1 and hand2
416     int highestValue1 = 0;
417     int highestValue2 = 0;
418
419     // calculate highest value in hand1
420     if ( numbers1[ 0 ] > 0 ) // hand1 contains Ace
421         highestValue1 = 13;
422     else
423     {
424         for ( int i = 1; i < numbers1.Length; i++ )
425         {
426             if ( numbers1[ i ] > 0 )
427             {
428                 highestValue1 = i;
429             } // end if
430         } // end for
431     } // end else
432
433     // calculate highest value in hand2
434     if ( numbers2[ 0 ] > 0 ) // hand2 contains Ace
435         highestValue2 = 13;
436     else
437     {
438         for ( int i = 1; i < numbers2.Length; i++ )
439         {
440             if ( numbers2[ i ] > 0 )
441             {
442                 highestValue2 = i;
443             } // end if
444         } // end for
445     } // end else
446
447     // compare and display result
448     if ( hand1Value > hand2Value )
449         Console.WriteLine( "\nResult: left hand is better" );
450     else if ( hand1Value < hand2Value )
451         Console.WriteLine( "\nResult: right hand is better" );
452     else
453     {
454         // test for the highest card
455         if ( highestValue1 > highestValue2 )
456             Console.WriteLine( "\nResult: left hand is better" );
457         else if ( highestValue1 < highestValue2 )
458             Console.WriteLine( "\nResult: right hand is better" );
459         else
460             Console.WriteLine( "\nResult: they are equal" );
461     } // end else
462 } // end method CompareTwoHands
463
464 // display hand values
465 public void DisplayHandValues()
466 {
467     string[] handValue = { "none", "none" };
```

```

468     int value = hand1Value;
469
470     for ( int i = 0; i < 2; i++ )
471     {
472         if ( i == 1 )
473             value = hand2Value;
474
475         switch ( value )
476         {
477             case 2:
478             case 3:
479                 handValue[ i ] = "One Pair";
480                 break;
481             case 4:
482             case 5:
483                 handValue[ i ] = "Two Pair";
484                 break;
485             case 6:
486             case 7:
487                 handValue[ i ] = "Three of a Kind";
488                 break;
489             case 8:
490             case 9:
491                 handValue[ i ] = "Straight";
492                 break;
493             case 10:
494             case 11:
495                 handValue[ i ] = "Full House";
496                 break;
497             case 12:
498             case 13:
499                 handValue[ i ] = "Flush";
500                 break;
501             case 14:
502             case 15:
503                 handValue[ i ] = "Four of a Kind";
504                 break;
505             case 16:
506                 handValue[ i ] = "Straight Flush";
507                 break;
508         } // end switch
509     } // end for
510
511     Console.Write( "{0,-20}", handValue[ 0 ] );
512     Console.WriteLine( "{0,-20}", handValue[ 1 ] );
513 } // end method DisplayHandValues
514 } // end class DeckOfCards

```

```

1 // Exercise 8.30 Solution: DeckOfCardsTest.cs
2 // Card shuffling and dealing application.
3 using System;
4

```

```

5 public class DeckOfCardsTest
6 {
7     // execute application
8     public static void Main( string[] args )
9     {
10         DeckOfCards myDeckOfCards = new DeckOfCards();
11         myDeckOfCards.Shuffle(); // place Cards in random order
12
13         Card[] hand1 = new Card[ 5 ]; // store first hand
14         Card[] hand2 = new Card[ 5 ]; // store second hand
15
16         // get first five cards
17         for ( int i = 0; i < 5; i++ )
18         {
19             hand1[ i ] = myDeckOfCards.DealCard(); // get next card
20             hand2[ i ] = myDeckOfCards.DealCard(); // get next card
21         } // end for
22
23         // display hand1 and hand2
24         Console.WriteLine( "{0,-20}{1,-20}", "Left hand:", "Right hand:" );
25
26         for ( int i = 0; i < hand1.Length; i++ )
27             Console.WriteLine( "{0,-20}{1,-20}", hand1[ i ], hand2[ i ] );
28
29         // display result
30         Console.WriteLine( "\nHand Values:" );
31         myDeckOfCards.CompareTwoHands( hand1, hand2 ); // compare two hands
32     } // end Main
33 } // end class DeckOfCardsTest

```

Left hand:	Right hand:
Queen of Hearts	Three of Hearts
Four of Diamonds	King of Spades
Three of Spades	Queen of Spades
Jack of Hearts	Eight of Clubs
Eight of Spades	Queen of Diamonds

Hand Values:
none One Pair

Result: right hand is better

Left hand:	Right hand:
Ace of Diamonds	Three of Hearts
Queen of Clubs	King of Hearts
Six of Hearts	King of Spades
Four of Hearts	Three of Spades
Ten of Hearts	King of Diamonds

Hand Values:
none Full House

Result: right hand is better

Left hand:	Right hand:
Deuce of Clubs	Seven of Hearts
Four of Clubs	Five of Diamonds
Jack of Diamonds	Nine of Spades
Six of Hearts	Three of Diamonds
Queen of Diamonds	Ten of Hearts
Hand Values:	
none	none
Result: left hand is better	

Special Section: Building Your Own Computer

In the next several problems, we take a temporary diversion from the world of high-level language programming to “peel open” a computer and look at its internal structure. We introduce machine-language programming and write several machine-language programs. To make this an especially valuable experience, we then build a computer (through the technique of software-based simulation) on which you can execute your machine-language programs.

8.31 (*Machine-Language Programming*) Let us create a computer called the Simpletron. As its name implies, it is a simple, but powerful, machine. The Simpletron runs programs written in the only language it directly understands: Simpletron Machine Language, or SML for short.

The Simpletron contains an *accumulator*—a special register in which information is put before the Simpletron uses that information in calculations or examines it in various ways. All the information in the Simpletron is handled in terms of *words*. A word is a signed four-digit decimal number, such as +3364, -1293, +0007 and -0001. The Simpletron is equipped with a 100-word memory, and these words are referenced by their location numbers 00, 01, ..., 99.

Before running an SML program, we must *load*, or place, the program into memory. The first instruction (or statement) of every SML program is always placed in location 00. The simulator will start executing at this location.

Each instruction written in SML occupies one word of the Simpletron’s memory (hence, instructions are signed four-digit decimal numbers). We shall assume that the sign of an SML instruction is always plus, but the sign of a data word may be either plus or minus. Each location in the Simpletron’s memory may contain an instruction, a data value used by a program or an unused (and hence undefined) area of memory. The first two digits of each SML instruction are the *operation code* specifying the operation to be performed. SML operation codes are summarized in Fig. 8.34.

Operation code	Meaning
<i>Input/output operations:</i>	
const int READ = 10;	Read a word from the keyboard into a specific location in memory.
const int WRITE = 11;	Write a word from a specific location in memory to the screen.

Fig. 8.29 | Simpletron Machine Language (SML) operation codes. (Part I of 2.)

Operation code	Meaning
<i>Load/store operations:</i>	
const int LOAD = 20;	Load a word from a specific location in memory into the accumulator.
const int STORE = 21;	Store a word from the accumulator into a specific location in memory.
<i>Arithmetic operations:</i>	
const int ADD = 30;	Add a word from a specific location in memory to the word in the accumulator (leave the result in the accumulator).
const int SUBTRACT = 31;	Subtract a word from a specific location in memory from the word in the accumulator (leave the result in the accumulator).
const int DIVIDE = 32;	Divide a word from a specific location in memory into the word in the accumulator (leave result in the accumulator).
const int MULTIPLY = 33;	Multiply a word from a specific location in memory by the word in the accumulator (leave the result in the accumulator).
<i>Transfer of control operations:</i>	
const int BRANCH = 40;	Branch to a specific location in memory.
const int BRANCHNEG = 41;	Branch to a specific location in memory if the accumulator is negative.
const int BRANCHZERO = 42;	Branch to a specific location in memory if the accumulator is zero.
const int HALT = 43;	Halt. The program has completed its task.

Fig. 8.29 | Simpletron Machine Language (SML) operation codes. (Part 2 of 2.)

The last two digits of an SML instruction are the *operand*—the address of the memory location containing the word to which the operation applies. Let's consider several simple SML programs.

The first SML program (Fig. 8.35) reads two numbers from the keyboard and computes and displays their sum. The instruction +1007 reads the first number from the keyboard, then places it into location 07 (which has been initialized to 0). Then instruction +1008 reads the next number into location 08. The *load* instruction, +2007, puts the first number into the accumulator, and the *add* instruction, +3008, adds the second number to the number in the accumulator. *All SML arithmetic instructions leave their results in the accumulator.* The *store* instruction, +2109, places the result in memory location 09, from which the *write* instruction, +1109, takes the number and displays it (as a signed four-digit decimal number). The *halt* instruction, +4300, terminates execution.

Location	Number	Instruction
00	+1007	(Read A)
01	+1008	(Read B)
02	+2007	(Load A)
03	+3008	(Add B)
04	+2109	(Store C)
05	+1109	(Write C)
06	+4300	(Halt)
07	+0000	(Variable A)
08	+0000	(Variable B)
09	+0000	(Result C)

Fig. 8.30 | SML program that reads two integers and computes their sum.

The second SML program (Fig. 8.16) reads two numbers from the keyboard and determines and displays the larger value. Note the use of the instruction +4107 as a conditional transfer of control, much the same as C#'s `if` statement.

Location	Number	Instruction
00	+1009	(Read A)
01	+1010	(Read B)
02	+2009	(Load A)
03	+3110	(Subtract B)
04	+4107	(Branch negative to 07)
05	+1109	(Write A)
06	+4300	(Halt)
07	+1110	(Write B)
08	+4300	(Halt)
09	+0000	(Variable A)
10	+0000	(Variable B)

Fig. 8.31 | SML program that reads two integers and determines the larger.

Now write SML programs to accomplish each of the following tasks:

- Use a sentinel-controlled loop to read positive numbers and compute and print their sum. Terminate input when a negative number is entered.

ANS:

```

00 +1009 (Read Value)
01 +2009 (Load Value)
02 +4106 (Branch negative to 06)
03 +3008 (Add Sum)
04 +2108 (Store Sum)

```

```

05 +4000 (Branch 00)
06 +1108 (Write Sum)
07 +4300 (Halt)
08 +0000 (Storage for Sum)
09 +0000 (Storage for Value)

```

- b) Use a counter-controlled loop to read seven numbers, some positive and some negative, then compute and display their average.

ANS:

```

00 +2018 (Load Counter)
01 +3121 (Subtract Termination)
02 +4211 (Branch zero to 11)
03 +2018 (Load Counter)
04 +3019 (Add Increment)
05 +2118 (Store Counter)
06 +1017 (Read Value)
07 +2016 (Load Sum)
08 +3017 (Add Value)
09 +2116 (Store Sum)
10 +4000 (Branch 00)
11 +2016 (Load Sum)
12 +3218 (Divide Counter)
13 +2120 (Store Result)
14 +1120 (Write Result)
15 +4300 (Halt)
16 +0000 (Variable Sum)
17 +0000 (Variable Value)
18 +0000 (Variable Counter)
19 +0001 (Variable Increment)
20 +0000 (Variable Result)
21 +0007 (Variable Termination)

```

- c) Read a series of numbers, then determine and display the largest number. The first number read indicates how many numbers should be processed.

ANS:

```

00 +1017 (Read EndValue)
01 +2018 (Load Counter)
02 +3117 (Subtract Endvalue)
03 +4215 (Branch zero to 15)
04 +2018 (Load Counter)
05 +3021 (Add Increment)
06 +2118 (Store Counter)
07 +1019 (Read Value)
08 +2020 (Load Largest)
09 +3119 (Subtract Value)
10 +4112 (Branch negative to 12)
11 +4001 (Branch 01)
12 +2019 (Load Value)
13 +2120 (Store Largest)
14 +4001 (Branch 01)
15 +1120 (Write Largest)
16 +4300 (Halt)
17 +0000 (Variable EndValue)
18 +0000 (Variable Counter)
19 +0000 (Variable Value)
20 +0000 (Variable Largest)
21 +0001 (Variable Increment)

```

8.32 (*Computer Simulator*) In this problem, you are going to build your own computer. No, you'll not be soldering components together. Rather, you'll use the powerful technique of *software-*

based simulation to create an object-oriented *software model* of the Simpletron of Exercise 8.31. Your Simpletron simulator will turn the computer you are using into a Simpletron, and you'll actually be able to run, test and debug the SML programs you wrote in Exercise 8.31.

When you run your Simpletron simulator, it should begin by displaying:

```
*** Welcome to Simpletron! ***
*** Please enter your program one instruction ***
*** ( or data word ) at a time into the input ***
*** text field. I will display the location ***
*** number and a question mark (?). You then ***
*** type the word for that location. Enter ***
*** -99999 to stop entering your program. ***
```

Your application should simulate the memory of the Simpletron with one-dimensional array memory of 100 elements. Now assume that the simulator is running, and let us examine the dialog as we enter the program of Fig. 8.36 (Exercise 8.31):

```
00 ? +1009
01 ? +1010
02 ? +2009
03 ? +3110
04 ? +4107
05 ? +1109
06 ? +4300
07 ? +1110
08 ? +4300
09 ? +0000
10 ? +0000
11 ? -99999
```

Your application should display the memory location followed by a question mark. Each of the values to the right of a question mark is input by the user. When the sentinel value -99999 is input, the application should display the following:

```
*** Program loading completed ***
*** Program execution begins ***
```

The SML program has now been placed (or loaded) in array memory. Now the Simpletron executes the SML program. Execution begins with the instruction in location 00 and, as in C#, continues sequentially, unless directed to some other part of the program by a transfer of control.

Use variable `accumulator` to represent the accumulator register. Use variable `instructionCounter` to keep track of the location in memory that contains the instruction being performed. Use variable `operationCode` to indicate the operation currently being performed (i.e., the left two digits of the instruction word). Use variable `operand` to indicate the memory location on which the current instruction operates. Thus, `operand` is the rightmost two digits of the instruction currently being performed. Do not execute instructions directly from memory. Rather, transfer the next instruction to be performed from memory to a variable called `instructionRegister`. Then “pick off” the left two digits and place them in `operationCode`, and “pick off” the right two digits and place them in `operand`. When the Simpletron begins execution, the special registers are all initialized to zero.

Now, let us “walk through” execution of the first SML instruction, +1009 in memory location 00. This procedure is called an *instruction execution cycle*.

The `instructionCounter` tells us the location of the next instruction to be performed. We *fetch* the contents of that location from memory by using the C# statement

```
instructionRegister = memory[ instructionCounter ];
```

The operation code and the operand are extracted from the instruction register by the statements


```
operationCode = instructionRegister / 100;
operand = instructionRegister % 100;
```

Now the Simpletron must determine that the operation code is actually a *read* (versus a *write*, a *load*, etc.). A switch differentiates among the 12 operations of SML. In the switch statement, the behavior of various SML instructions is simulated as shown in Fig. 8.37. We discuss branch instructions shortly and leave the others to you.

Instruction	Description
<i>read:</i>	Display the prompt "Enter an integer", then input the integer and store it in location memory[operand].
<i>load:</i>	accumulator = memory[operand];
<i>add:</i>	accumulator += memory[operand];
<i>halt:</i>	This instruction displays the message *** Simpletron execution terminated ***

Fig. 8.32 | Behavior of several SML instructions in the Simpletron.

When the SML program completes execution, the name and contents of each register, as well as the complete contents of memory, should be displayed. Such a printout is often called a memory dump. To help you program your dump method, a sample dump format is shown in Fig. 8.38. Note that a dump after executing a Simpletron program would show the actual values of instructions and data values at the moment execution terminated.

```
REGISTERS:
accumulator      +0000
instructionCounter 00
instructionRegister +0000
operationCode     00
operand          00

MEMORY:
 0 0 +0000 +0000 +0000 +0000 +0000 +0000 +0000 +0000 +0000
10 0 +0000 +0000 +0000 +0000 +0000 +0000 +0000 +0000 +0000
20 0 +0000 +0000 +0000 +0000 +0000 +0000 +0000 +0000 +0000
30 0 +0000 +0000 +0000 +0000 +0000 +0000 +0000 +0000 +0000
40 0 +0000 +0000 +0000 +0000 +0000 +0000 +0000 +0000 +0000
50 0 +0000 +0000 +0000 +0000 +0000 +0000 +0000 +0000 +0000
60 0 +0000 +0000 +0000 +0000 +0000 +0000 +0000 +0000 +0000
70 0 +0000 +0000 +0000 +0000 +0000 +0000 +0000 +0000 +0000
80 0 +0000 +0000 +0000 +0000 +0000 +0000 +0000 +0000 +0000
90 0 +0000 +0000 +0000 +0000 +0000 +0000 +0000 +0000 +0000
```

Fig. 8.33 | A sample memory dump.

Let us proceed with the execution of our program's first instruction—namely, the +1009 in location 00. As we have indicated, the switch statement simulates this task by prompting the user to enter a value, reading the value and storing it in memory location memory[operand]. The value is then read into location 09.

At this point, simulation of the first instruction is completed. All that remains is to prepare the Simpletron to execute the next instruction. Since the instruction just performed was not a transfer of control, we need merely increment the instruction-counter register as follows:

```
++instructionCounter;
```

This action completes the simulated execution of the first instruction. The entire process (i.e., the instruction execution cycle) begins anew with the fetch of the next instruction to execute.

Now let us consider how the branching instructions—the transfers of control—are simulated. All we need to do is adjust the value in the instruction counter appropriately. Therefore, the unconditional branch instruction (40) is simulated within the switch as

```
instructionCounter = operand;
```

The conditional “branch if accumulator is zero” instruction is simulated as

```
if ( accumulator == 0 )
    instructionCounter = operand;
```

At this point, you should implement your Simpletron simulator and run each of the SML programs you wrote in Exercise 8.31. If you desire, you may embellish SML with additional features and provide for these features in your simulator.

Your simulator should check for various types of errors. During the program-loading phase, for example, each number the user types into the Simpletron’s memory must be in the range -9999 to +9999. Your simulator should test that each number entered is in this range and, if not, keep prompting the user to re-enter the number until the user enters a correct number.

During the execution phase, your simulator should check for various serious errors, such as attempts to divide by zero, attempts to execute invalid operation codes and accumulator overflows (i.e., arithmetic operations resulting in values larger than +9999 or smaller than -9999). Such serious errors are called *fatal errors*. When a fatal error is detected, your simulator should display an error message such as

```
*** Attempt to divide by zero ***
*** Simpletron execution abnormally terminated ***
```

and should display a full computer dump in the format we discussed previously. This treatment will help the user locate the error in the program.

ANS:

```
1 // Exercise 8.32 Solution: Simulator.cs
2 // A computer simulator
3 using System;
4
5 public class Simulator
6 {
7     // list of SML instructions
8     const int READ = 10;
9     const int WRITE = 11;
10    const int LOAD = 20;
11    const int STORE = 21;
12    const int ADD = 30;
13    const int SUBTRACT = 31;
14    const int DIVIDE = 32;
15    const int MULTIPLY = 33;
16    const int BRANCH = 40;
17    const int BRANCH_NEG = 41;
```

```

18  const int BRANCH_ZERO = 42;
19  const int HALT = 43;
20
21  static int accumulator; // accumulator register
22  static int instructionCounter; // instruction counter, a memory address
23  static int operand; // argument for the operator
24  static int operationCode; // determines the operation
25  static int instructionRegister; // register holding the SML instruction
26
27  static int[] memory; // simpletron memory
28  static int index = 0; // number of instructions entered in memory
29
30  // runs the simpletron simulator, reads instructions and executes
31  public static void Main( string[] args )
32  {
33      // initialize the registers
34      InitializeRegisters();
35
36      // prompt the user to enter instructions
37      PrintInstructions();
38      LoadInstructions();
39
40      // execute the program and display the memory dump when finished
41      Execute();
42      Dump();
43  } // end method Main
44
45  // set all registers to the correct start value
46  public static void InitializeRegisters()
47  {
48      memory = new int[ 100 ];
49      accumulator = 0;
50      instructionCounter = 0;
51      instructionRegister = 0;
52      operand = 0;
53      operationCode = 0;
54
55      for ( int k = 0; k < memory.Length; k++ )
56          memory[ k ] = 0;
57  } // end method InitializeRegisters
58
59  // display out user instructions
60  public static void PrintInstructions()
61  {
62      Console.WriteLine( "{0}\n{1}\n{2}\n{3}\n{4}\n{5}\n{6}",
63          "*** Welcome to Simpletron! ***",
64          "*** Please enter your program one instruction ***",
65          "*** ( or data word ) at a time into the input ***",
66          "*** text field. I will display the location ***",
67          "*** number and a question mark (?). You then ***",
68          "*** type the word for that location. Enter ***",
69          "*** -99999 to stop entering your program. ***" );
70  } // end method PrintInstructions
71

```

```

72 // read in user input, test it, perform operations
73 public static void LoadInstructions()
74 {
75     Console.Write( "{0:D2} ? ", index );
76     int instruction = Convert.ToInt32( Console.ReadLine() );
77
78     while ( instruction != -9999 && index < 100 )
79     {
80         if ( Validate( instruction ) )
81             memory[ index++ ] = instruction;
82         else
83             Console.WriteLine( "Input invalid." );
84
85         Console.Write( "{0:D2} ? ", index );
86         instruction = Convert.ToInt32( Console.ReadLine() );
87     } // end while
88
89     Console.WriteLine( "*** Program loading completed ***" );
90 } // end method LoadInstructions
91
92 // ensure value is within range
93 // returns true if the value is within range, otherwise returns false
94 public static bool Validate( int value )
95 {
96     return ( -9999 <= value ) && ( value <= 9999 );
97 } // end method Validate
98
99 // ensure that accumulator has not overflowed
100 public static bool TestOverflow()
101 {
102     if ( !Validate( accumulator ) )
103     {
104         Console.WriteLine(
105             "*** Fatal error. Accumulator overflow. ***" );
106         return true;
107     } // end if
108
109     return false;
110 } // end method TestOverflow
111
112 // perform all simulator functions
113 public static void Execute()
114 {
115     Console.WriteLine( "*** Program execution begins ***" );
116
117     // continue executing until we reach the end of the program
118     // it is possible that the program can terminate beforehand though
119     while ( instructionCounter < index )
120     {
121         // read the instruction into the registers
122         instructionRegister = memory[ instructionCounter ];
123         operationCode = instructionRegister / 100;
124         operand = instructionRegister % 100;
125     }

```

```

126      // go to next instruction, this will only be overridden
127      // by the branch commands
128      ++instructionCounter;
129
130      switch ( operationCode )
131      {
132          case READ:
133              // read an integer
134              Console.Write( "Enter an integer: " );
135              memory[ operand ] = Convert.ToInt32( Console.ReadLine() );
136              break;
137          case WRITE:
138              // outputs the contents of a memory address
139              Console.WriteLine( "Contents of {0,2} is {1}",
140                  operand, memory[ operand ] );
141              break;
142          case LOAD:
143              // load a memory address into the accumulator
144              accumulator = memory[ operand ];
145              break;
146          case STORE:
147              // store the contents of the accumulator to an address
148              memory[ operand ] = accumulator;
149              break;
150          case ADD:
151              // adds the contents of an address to the accumulator
152              accumulator += memory[ operand ];
153
154              if ( TestOverflow() )
155                  return;
156
157              break;
158          case SUBTRACT:
159              // subtracts the contents of an address from the
160              // accumulator
161              accumulator -= memory[ operand ];
162
163              if ( TestOverflow() )
164                  return;
165
166              break;
167          case MULTIPLY:
168              // multiplies the accumulator with the contents of an
169              // address
170              accumulator *= memory[ operand ];
171
172              if ( TestOverflow() )
173                  return;
174
175              break;
176          case DIVIDE:
177              // divides the accumulator by the contents of an address
178              if ( memory[ operand ] == 0 )
179                  {

```

```

180         Console.WriteLine(
181             "*** Fatal error. Attempt to divide by zero. ***" );
182         return;
183     } // end if
184
185     accumulator /= memory[ operand ];
186     break;
187 case BRANCH:
188     // jumps to an address
189     instructionCounter = operand;
190     break;
191 case BRANCH_NEG:
192     // jumps to an address if the accumulator is negative
193     if ( accumulator < 0 )
194         instructionCounter = operand;
195
196     break;
197 case BRANCH_ZERO:
198     // jumps to an address if the accumulator is zero
199     if ( accumulator == 0 )
200         instructionCounter = operand;
201
202     break;
203 case HALT:
204     // terminates execution
205     Console.WriteLine(
206         "*** Simpletron execution terminated ***" );
207     return;
208 default:
209     // all other cases are not valid opcodes
210     Console.WriteLine(
211         "*** Fatal error. Invalid operation code. ***" );
212     return;
213 } // end switch
214 } // end while
215 } // end method Execute
216
217 // displays the values of the registers
218 public static void DisplayRegisters()
219 {
220     Console.WriteLine( "REGISTERS:" );
221     Console.WriteLine( "{0,-24}{1:D5}", "Accumulator:", accumulator );
222     Console.WriteLine( "{0,-27}{1:D2}", "InstructionCounter:",
223         instructionCounter );
224     Console.WriteLine( "{0,-24}{1:D5}", "InstructionRegister:",
225         instructionRegister );
226     Console.WriteLine( "{0,-27}{1:D2}", "OperationCode:",
227         operationCode );
228     Console.WriteLine( "{0,-27}{1:D2}", "Operand:", operand );
229 } // end method DisplayRegisters
230
231 // output memory information
232 public static void Dump()
233 {

```

```

234     DisplayRegisters();
235
236     Console.WriteLine( "\nMEMORY:" );
237
238     // display column headings
239     Console.Write( "  " );
240
241     for ( int k = 0; k < 10; k++ )
242         Console.Write( "{0,7:D2}", k );
243
244     Console.WriteLine();
245
246     // display the memory dump
247     for ( int k = 0; k < 10; k++ )
248     {
249         // display the row label
250         Console.Write( "{0:D2}", k * 10 );
251
252         // display the contents of each memory locations
253         for ( int i = 0; i < 10; i++ )
254             Console.Write( " {0:D5}", memory[ k * 10 + i ] );
255
256         Console.WriteLine();
257     } // end for
258 } // end method Dump
259 } // end class Simulator

```

```

*** Welcome to Simpletron! ***
*** Please enter your program one instruction ***
*** ( or data word ) at a time into the input ***
*** text field. I will display the location ***
*** number and a question mark (?). You then ***
*** type the word for that location. Enter ***
*** -99999 to stop entering your program. ***
00 ? 1007
01 ? 1008
02 ? 2007
03 ? 3008
04 ? 2109
05 ? 1109
06 ? 4300
07 ? 0000
08 ? 0000
09 ? 0000
10 ? -99999
*** Program loading completed ***
*** Program execution begins ***
Enter an integer: 10
Enter an integer: 3
Contents of 9 is 13
*** Simpletron execution terminated ***

```

REGISTERS:

Accumulator: 00013
 InstructionCounter: 07
 InstructionRegister: 04300
 OperationCode: 43
 Operand: 00

MEMORY:

	00	01	02	03	04	05	06	07	08	09
00	01007	01008	02007	03008	02109	01109	04300	00010	00003	00013
10	00000	00000	00000	00000	00000	00000	00000	00000	00000	00000
20	00000	00000	00000	00000	00000	00000	00000	00000	00000	00000
30	00000	00000	00000	00000	00000	00000	00000	00000	00000	00000
40	00000	00000	00000	00000	00000	00000	00000	00000	00000	00000
50	00000	00000	00000	00000	00000	00000	00000	00000	00000	00000
60	00000	00000	00000	00000	00000	00000	00000	00000	00000	00000
70	00000	00000	00000	00000	00000	00000	00000	00000	00000	00000
80	00000	00000	00000	00000	00000	00000	00000	00000	00000	00000
90	00000	00000	00000	00000	00000	00000	00000	00000	00000	00000

8.33 (*Project: Simpletron Simulator Modifications*) [Note: No solutions are provided in the *Instructor's Manual*.] In Exercise 8.32, you wrote a software simulation of a computer that executes programs written in Simpletron Machine Language (SML). In this exercise, we propose several modifications and enhancements to the Simpletron Simulator.

- Extend the Simpletron Simulator's memory to contain 1000 memory locations to enable the Simpletron to handle larger programs.
- Allow the simulator to perform remainder calculations. This modification requires an additional SML instruction.
- Allow the simulator to perform exponentiation calculations. This modification requires an additional SML instruction.
- Modify the simulator to use hexadecimal values rather than integer values to represent SML instructions.
- Modify the simulator to allow output of a newline. This modification requires an additional SML instruction.
- Modify the simulator to process floating-point values in addition to integer values.
- Modify the simulator to handle string input. [*Hint*: Each Simpletron word can be divided into two groups, each holding a two-digit integer. Each two-digit integer represents the ASCII (see Appendix Fy) decimal equivalent of a character. Add a machine-language instruction that will input a string and store the string beginning at a specific Simpletron memory location. The first half of the word at that location will be a count of the number of characters in the string (i.e., the length of the string). Each succeeding half-word contains one ASCII character expressed as two decimal digits. The machine-language instruction converts each character into its ASCII equivalent and assigns it to a half-word.]
- Modify the simulator to handle output of strings stored in the format of *Part g*. [*Hint*: Add a machine-language instruction that will display a string beginning at a certain Simpletron memory location. The first half of the word at that location is a count of the number of characters in the string (i.e., the length of the string). Each succeeding half-word contains one ASCII character expressed as two decimal digits. The machine-language instruction checks the length and displays the string by translating each two-digit number into its equivalent character.]

Making a Difference Exercise

8.34 (Polling) The Internet and the web are enabling more people to network, join a cause, voice opinions, and so on. The presidential candidates in 2008 used the Internet intensively to get out their messages and raise money for their campaigns. In this exercise, you'll write a simple polling program that allows users to rate five social-consciousness issues from 1 (least important) to 10 (most important). Pick five causes that are important to you (for example, political issues, global environmental issues). Use a one-dimensional array `topics` (of type `String`) to store the five causes. To summarize the survey responses, use a 5-row, 10-column two-dimensional array `responses` (of type `Integer`), each row corresponding to an element in the `topics` array. When the program runs, it should ask the user to rate each issue. Have your friends and family respond to the survey. Then have the program display a summary of the results, including:

- A tabular report with the five topics down the left side and the 10 ratings across the top, listing in each column the number of ratings received for each topic.
- To the right of each row, show the average of the ratings for that issue.
- Which issue received the highest point total? Display both the issue and the point total.
- Which issue received the lowest point total? Display both the issue and the point total.

ANS:

```

1 // Exercise 8.34 Solution: Poll.cs
2 // Analyze polling data on various issues.
3 using System;
4
5 class Poll
6 {
7     // constants
8     static string[] topics = { "global warming", "the economy",
9                               "war", "health care", "education" };
10
11     public static void Main( string[] args )
12     {
13         const int topicCount = 5;
14         const int maxRating = 10;
15
16         // empty initialization list initializes array to zero
17         int[ , ] responses = new int[ topicCount, maxRating ];
18         int choice = 1; // sentinel to decide when to exit loop
19
20         while ( choice != 0 )
21         {
22             Console.WriteLine(); // add extra blank line
23             PollUser( responses );
24
25             // see if we should stop getting user input
26             Console.Write( "Enter more data? (1=yes, 0=no): " );
27             choice = Convert.ToInt32( Console.ReadLine() );
28         } // end while
29
30         DisplayResults( responses );
31     } // end Main
32

```

```

33 // get ratings on topics from one user
34 public static void PollUser( int[,] responses )
35 {
36     for ( int i = 0; i < responses.GetLength( 0 ); ++i )
37     {
38         Console.WriteLine(
39             "On a scale of 1-{0}, how important is {1}?",
40             responses.GetLength( 1 ), topics[ i ] ); // ask question
41         int rating; // rating user gave for this
42
43         do
44         {
45             Console.Write( "> " ); // display prompt
46             rating = Convert.ToInt32( Console.ReadLine() );
47         } // end do
48         while ( rating < 1 || rating > responses.GetLength( 1 ) );
49
50         ++responses[ i, rating - 1 ]; // store rating
51     } // end for
52 } // end function PollUser
53
54 // display polling results in tabular format
55 public static void DisplayResults( int[,] responses )
56 {
57     // display table header
58     Console.Write( "\n{0, -15}", "Topic" );
59
60     for ( int i = 1; i <= responses.GetLength( 1 ); ++i )
61         Console.Write( "{0, 4}", i );
62
63     Console.WriteLine( "{0, 10}", "Average" );
64
65     // display rating counts and averages for each topic
66     for ( int i = 0; i < responses.GetLength( 0 ); ++i )
67     {
68         Console.Write( "{0, -15}", topics[ i ] ); // display topic
69
70         // display number of times topic was given this score
71         for ( int j = 0; j < responses.GetLength( 1 ); ++j )
72             Console.Write( "{0, 4}", responses[ i, j ] );
73
74         // display average rating for this topic
75         Console.WriteLine( "{0, 10:F1}",
76             CalculateAverage( responses, i ) );
77     } // end for
78
79     Console.WriteLine(); // add blank line
80     DisplayHighest( responses ); // display highest-rated issue
81     DisplayLowest( responses ); // display lowest-rated issue
82 } // end function DisplayResults
83
84 // calculate average number of points
85 public static double CalculateAverage( int[,] votes, int row )
86 {

```

```

87     int count = 0; // total number of votes
88
89     for ( int i = 0; i < votes.GetLength( 1 ); ++i )
90         count += votes[ row, i ]; // add number of responses
91
92     // return average
93     return ( double ) CountPoints( votes, row ) / count;
94 } // end function CalculateAverage
95
96 // display topic with most points
97 public static void DisplayHighest( int[,] responses )
98 {
99     int max = CountPoints( responses, 0 ); // maximum number of points
100    int maxIndex = 0; // index of issue with maximum number of points
101
102    for ( int i = 1; i < responses.GetLength( 0 ); ++i )
103    {
104        if ( CountPoints( responses, i ) > max )
105        {
106            // larger point count found, update maximum value and index
107            max = CountPoints( responses, i );
108            maxIndex = i;
109        } // end if
110    } // end for
111
112    Console.WriteLine( "Highest points: {0} ({1})\n",
113        topics[ maxIndex ], max );
114 } // end function DisplayHighest
115
116 // display topic with fewest points
117 public static void DisplayLowest( int[,] responses )
118 {
119     int min = CountPoints( responses, 0 ); // minimum number of points
120     int minIndex = 0; // index of issue with minimum number of points
121
122     for ( int i = 1; i < responses.GetLength( 0 ); ++i )
123     {
124         if ( CountPoints( responses, i ) < min )
125         {
126             // smaller point count found, update minimum value and index
127             min = CountPoints( responses, i );
128             minIndex = i;
129         } // end if
130     } // end for
131
132     Console.WriteLine( "Lowest points: {0} ({1})\n",
133         topics[ minIndex ], min );
134 } // end function DisplayLowest
135
136 // calculate total number of points for a given topic
137 public static int CountPoints( int[,] votes, int row )
138 {
139     int sum = 0; // total number of votes
140

```

```

141     for ( int i = 0; i < votes.GetLength( 1 ); ++i )
142         sum += votes[row, i ] * ( i + 1 ); // add weighted count
143
144     return sum;
145 } // end function CountPoints
146 } // end class Poll

```

```

On a scale of 1-10, how important is global warming?
> 8
On a scale of 1-10, how important is the economy?
> 10
On a scale of 1-10, how important is war?
> 9
On a scale of 1-10, how important is health care?
> 7
On a scale of 1-10, how important is education?
> 8
Enter more data? (1=yes, 0=no): 1
On a scale of 1-10, how important is global warming?
> 2
On a scale of 1-10, how important is the economy?
> 4
On a scale of 1-10, how important is war?
> 8
On a scale of 1-10, how important is health care?
> 6
On a scale of 1-10, how important is education?
> 9
Enter more data? (1=yes, 0=no): 1
On a scale of 1-10, how important is global warming?
> 3
On a scale of 1-10, how important is the economy?
> 7
On a scale of 1-10, how important is war?
> 5
On a scale of 1-10, how important is health care?
> 6
On a scale of 1-10, how important is education?
> 9
Enter more data? (1=yes, 0=no): 1
On a scale of 1-10, how important is global warming?
> 10
On a scale of 1-10, how important is the economy?
> 2
On a scale of 1-10, how important is war?
> 3
On a scale of 1-10, how important is health care?
> 6
On a scale of 1-10, how important is education?
> 8
Enter more data? (1=yes, 0=no): 1
On a scale of 1-10, how important is global warming?
> 1
On a scale of 1-10, how important is the economy?
> 2

```

On a scale of 1-10, how important is war?

> 3

On a scale of 1-10, how important is health care?

> 4

On a scale of 1-10, how important is education?

> 5

Enter more data? (1=yes, 0=no): 0


Topic	1	2	3	4	5	6	7	8	9	10	Average
global warming	1	1	1	0	0	0	0	1	0	1	4.8
the economy	0	2	0	1	0	0	1	0	0	1	5.0
war	0	0	2	0	1	0	0	1	1	0	5.6
health care	0	0	0	1	0	3	1	0	0	0	5.8
education	0	0	0	0	1	0	0	2	2	0	7.8

Highest points: education (39)

Lowest points: global warming (24)

Introduction to LINQ and the **List** Collection

9



*To write it, it took three months;
to conceive it three minutes; to
collect the data in it—all my
life.*

—F. Scott Fitzgerald

*Science is feasible when the
variables are few and can be
enumerated...*

—Paul Valéry

*You shall listen to all sides and
filter them from your self.*

—Walt Whitman

*The portraitist can select one
tiny aspect of everything shown
at a moment to incorporate into
the final painting.*

—Robert Nozick

List, list, O, list!

—William Shakespeare

Objectives

In this chapter you'll learn:

- Basic LINQ concepts.
- How to query an array using LINQ.
- Basic .NET collections concepts.
- How to create and use a generic **List** collection.
- How to query a generic **List** collection using LINQ.

Self-Review Exercises

- 9.1** Fill in the blanks in each of the following statements:
- a) Use the _____ property of the `List` class to find the number of elements in the `List`.
ANS: `Count`.
 - b) The LINQ _____ clause is used for filtering.
ANS: `where`.
 - c) _____ are classes specifically designed to store groups of objects and provide methods that organize, store and retrieve those objects.
ANS: `Collections`.
 - d) To add an element to the end of a `List`, use the _____ method.
ANS: `Add`.
 - e) To get only unique results from a LINQ query, use the _____ method.
ANS: `Distinct`.
- 9.2** State whether each of the following is *true* or *false*. If *false*, explain why.
- a) The `orderby` clause in a LINQ query can sort only in ascending order.
ANS: *False*. The `descending` modifier is used to make `orderby` sort in descending order.
 - b) LINQ queries can be used on both arrays and collections.
ANS: *True*.
 - c) The `Remove` method of the `List` class removes an element at a specific index.
ANS: *False*. `Remove` removes the first element equal to its argument. `RemoveAt` removes the element at a specific index.

Exercises

- 9.3** (*Querying an Array of Invoice Objects*) Use the class `Invoice` provided in the `ex09_03` folder with this chapter's examples to create an array of `Invoice` objects. Use the sample data shown in Fig. 9.8. Class `Invoice` includes four properties—a `PartNumber` (type `int`), a `PartDescription` (type `string`), a `Quantity` of the item being purchased (type `int`) and a `Price` (type `decimal`). Perform the following queries on the array of `Invoice` objects and displays the results:
- a) Use LINQ to sort the `Invoice` objects by `PartDescription`.
 - b) Use LINQ to sort the `Invoice` objects by `Price`.
 - c) Use LINQ to select the `PartDescription` and `Quantity` and sort the results by `Quantity`.
 - d) Use LINQ to select from each `Invoice` the `PartDescription` and the value of the `Invoice` (i.e., `Quantity * Price`). Name the calculated column `InvoiceTotal`. Order the results by `Invoice` value. [*Hint:* Use `let` to store the result of `Quantity * Price` in a new range variable `total`.]
 - e) Using the results of the LINQ query in *Part d*, select the `InvoiceTotals` in the range \$200 to \$500.

Part number	Part description	Quantity	Price
83	Electric sander	7	57.98
24	Power saw	18	99.99
7	Sledge hammer	11	21.50

Fig. 9.8 | Sample data for Exercise 9.3.

Part number	Part description	Quantity	Price
77	Hammer	76	11.99
39	Lawn mower	3	79.50
68	Screwdriver	106	6.99
56	Jig saw	21	11.00
3	Wrench	34	7.50

Fig. 9.8 | Sample data for Exercise 9.3.

```

1 // Exercise 9.3 Solution: Invoice.cs
2 // Invoice class.
3 public class Invoice
4 {
5     // declare variables for Invoice object
6     private int quantityValue;
7     private decimal priceValue;
8
9     // auto-implemented property PartNumber
10    public int PartNumber { get; set; }
11
12    // auto-implemented property PartDescription
13    public string PartDescription { get; set; }
14
15    // four-argument constructor
16    public Invoice( int part, string description,
17                  int count, decimal pricePerItem )
18    {
19        PartNumber = part;
20        PartDescription = description;
21        Quantity = count;
22        Price = pricePerItem;
23    } // end constructor
24
25    // property for quantityValue; ensures value is positive
26    public int Quantity
27    {
28        get
29        {
30            return quantityValue;
31        } // end get
32        set
33        {
34            if ( value > 0 ) // determine whether quantity is positive
35                quantityValue = value; // valid quantity assigned
36        } // end set
37    } // end property Quantity
38
39    // property for pricePerItemValue; ensures value is positive
40    public decimal Price
41    {

```

```

42     get
43     {
44         return priceValue;
45     } // end get
46     set
47     {
48         if ( value >= 0M ) // determine whether price is non-negative
49             priceValue = value; // valid price assigned
50     } // end set
51 } // end property Price
52
53 // return string containing the fields in the Invoice in a nice format
54 public override string ToString()
55 {
56     // left justify each field, and give large enough spaces so
57     // all the columns line up
58     return string.Format( "{0,-5} {1,-20} {2,-5} {3,6:C}",
59         PartNumber, PartDescription, Quantity, Price );
60 } // end method ToString
61 } // end class Invoice

```

```

1  // Exercise 9.3 Solution: LINQInvoiceArray.cs
2  // Use LINQ to extract data from an array of Invoice objects.
3  using System;
4  using System.Linq;
5  using System.Collections.Generic;
6
7  public class LINQInvoiceArray
8  {
9      public static void Main( string[] args )
10     {
11         // initialize array of invoices
12         Invoice[] invoices = {
13             new Invoice( 83, "Electric sander", 7, 57.98M ),
14             new Invoice( 24, "Power saw", 18, 99.99M ),
15             new Invoice( 7, "Sledge hammer", 11, 21.5M ),
16             new Invoice( 77, "Hammer", 76, 11.99M ),
17             new Invoice( 39, "Lawn mower", 3, 79.5M ),
18             new Invoice( 68, "Screwdriver", 106, 6.99M ),
19             new Invoice( 56, "Jig saw", 21, 11M ),
20             new Invoice( 3, "Wrench", 34, 7.5M ) }; // end initializer list
21
22         // use LINQ to sort invoices by description
23         var sortedByDescription =
24             from item in invoices
25             orderby item.PartDescription
26             select item;
27
28         // display invoices, sorted by description
29         Console.WriteLine( "Sorted by description:" );
30         foreach ( var item in sortedByDescription )
31             Console.WriteLine( item );

```

32

```

33     // use LINQ to sort invoices by price
34     var sortedByPrice =
35         from item in invoices
36         orderby item.Price
37         select item;
38
39     // display invoices, sorted by price
40     Console.WriteLine( "\nSorted by price:" );
41     foreach ( var item in sortedByPrice )
42         Console.WriteLine( item );
43
44     // use LINQ to select description and quantity, sort by quantity
45     var descriptionAndQuantity =
46         from item in invoices
47         orderby item.Quantity
48         select new { item.PartDescription, item.Quantity };
49
50     // display description and quantity, sorted by quantity
51     Console.WriteLine(
52         "\nSelect description and quantity, sort by quantity:" );
53     foreach ( var item in descriptionAndQuantity )
54         Console.WriteLine( item );
55
56     // use LINQ to select description and calculated
57     // invoice total; sort by invoice total
58     var descriptionAndTotal =
59         from item in invoices
60         let total = item.Quantity * item.Price
61         orderby total
62         select new { item.PartDescription, InvoiceTotal = total };
63
64     // display description and calculated invoice total
65     Console.WriteLine(
66         "\nSelect description and invoice total, sort by invoice total:" );
67     foreach ( var item in descriptionAndTotal )
68         Console.WriteLine( item );
69
70     // use LINQ to filter previous query results on range of totals
71     var totalBetween200And500 =
72         from item in descriptionAndTotal
73         where item.InvoiceTotal > 200M && item.InvoiceTotal < 500M
74         select item;
75
76     // display filtered descriptions and invoice totals
77     Console.WriteLine( string.Format(
78         "\nInvoice totals between {0:C} and {1:C}:", 200, 500 ) );
79     foreach ( var item in totalBetween200And500 )
80         Console.WriteLine( item );
81
82     Console.WriteLine();
83 } // end Main
84 } // end class LINQInvoiceArray

```

Sorted by description:

83	Electric sander	7	\$57.98
77	Hammer	76	\$11.99
56	Jig saw	21	\$11.00
39	Lawn mower	3	\$79.50
24	Power saw	18	\$99.99
68	Screwdriver	106	\$6.99
7	Sledge hammer	11	\$21.50
3	Wrench	34	\$7.50

Sorted by price:

68	Screwdriver	106	\$6.99
3	Wrench	34	\$7.50
56	Jig saw	21	\$11.00
77	Hammer	76	\$11.99
7	Sledge hammer	11	\$21.50
83	Electric sander	7	\$57.98
39	Lawn mower	3	\$79.50
24	Power saw	18	\$99.99

Select description and quantity, sort by quantity:

```
{ PartDescription = Lawn mower, Quantity = 3 }
{ PartDescription = Electric sander, Quantity = 7 }
{ PartDescription = Sledge hammer, Quantity = 11 }
{ PartDescription = Power saw, Quantity = 18 }
{ PartDescription = Jig saw, Quantity = 21 }
{ PartDescription = Wrench, Quantity = 34 }
{ PartDescription = Hammer, Quantity = 76 }
{ PartDescription = Screwdriver, Quantity = 106 }
```

Select description and invoice total, sort by invoice total:

```
{ PartDescription = Jig saw, InvoiceTotal = 231 }
{ PartDescription = Sledge hammer, InvoiceTotal = 236.5 }
{ PartDescription = Lawn mower, InvoiceTotal = 238.5 }
{ PartDescription = Wrench, InvoiceTotal = 255.0 }
{ PartDescription = Electric sander, InvoiceTotal = 405.86 }
{ PartDescription = Screwdriver, InvoiceTotal = 740.94 }
{ PartDescription = Hammer, InvoiceTotal = 911.24 }
{ PartDescription = Power saw, InvoiceTotal = 1799.82 }
```

Invoice totals between \$200.00 and \$500.00:

```
{ PartDescription = Jig saw, InvoiceTotal = 231 }
{ PartDescription = Sledge hammer, InvoiceTotal = 236.5 }
{ PartDescription = Lawn mower, InvoiceTotal = 238.5 }
{ PartDescription = Wrench, InvoiceTotal = 255.0 }
{ PartDescription = Electric sander, InvoiceTotal = 405.86 }
```

9.4 (*Duplicate Word Removal*) Write a console application that inputs a sentence from the user (assume no punctuation), then determines and displays the nonduplicate words in alphabetical order. Treat uppercase and lowercase letters the same. [*Hint*: You can use `string` method `Split` with no arguments, as in `sentence.Split()`, to break a sentence into an array of strings containing the individual words. By default, `Split` uses spaces as delimiters. Use `string` method `ToLower` in the `select` and `orderby` clauses of your LINQ query to obtain the lowercase version of each word.]

ANS:

```

1 // Exercise 9.4 Solution: LINQDistinctWords.cs
2 // Prompt the user for a sentence and display the distinct words.
3 using System;
4 using System.Linq;
5
6 public class LINQDistinctWords
7 {
8     public static void Main( string[] args )
9     {
10         // prompt the user for input
11         Console.WriteLine( "Please enter a sentence:" );
12         string sentence = Console.ReadLine(); // read input sentence
13         string[] words = sentence.Split(); // split sentence into words
14
15         // use LINQ to sort the words and convert to lowercase
16         var sortedWords =
17             from word in words
18             let lowerCaseWord = word.ToLower()
19             orderby lowerCaseWord
20             select lowerCaseWord;
21
22         Console.WriteLine( "\nYou entered:" ); // display header
23         Console.WriteLine( sentence ); // display the input
24         Console.Write( "\nDistinct words:" ); // display header
25
26         // display only the distinct words
27         foreach ( var word in sortedWords.Distinct() )
28             Console.Write( " {0}", word );
29
30         Console.WriteLine(); // output end of line
31     } // end Main
32 } // end class LINQDistinctWords

```

Please enter a sentence:
the quick brown fox jumps over the lazy dog

You entered:
 the quick brown fox jumps over the lazy dog

Distinct words: brown dog fox jumps lazy over quick the

9.5 (*Sorting Letters and Removing Duplicates*) Write a console application that inserts 30 random letters into a `List< char >`. Perform the following queries on the `List` and display your results: [Hint: Strings can be indexed like arrays to access a character at a specific index.]

- Use LINQ to sort the `List` in ascending order.
- Use LINQ to sort the `List` in descending order.
- Display the `List` in ascending order with duplicates removed.

ANS:

```

1  // Exercise 9.5 Solution: LINQLetterList.cs
2  // Sort a List of char using LINQ.
3  using System;
4  using System.Linq;
5  using System.Collections.Generic;
6
7  public class LINQLetterList
8  {
9      public static void Main( string[] args )
10     {
11         List< char > theList = new List< char >(); // create character list
12         // create random number generator
13         Random randomNumber = new Random();
14
15         // we index into the alphabet to convert
16         // numbers 0-25 to letters a-z
17         string alphabet = "abcdefghijklmnopqrstuvwxyz";
18
19         // generate 30 random letters
20         for ( int i = 0; i < 30; i++ )
21         {
22             // generate number from 0 to 25 to
23             // select a letter from the string
24             int letterNumber = randomNumber.Next( 26 );
25
26             // convert the letter number to the actual letter
27             char letter = alphabet[ letterNumber ];
28             theList.Add( letter ); // add the letter to the List
29         } // end for
30
31         Console.WriteLine( "Generated List:" ); // display original List
32         foreach ( var item in theList )
33             Console.Write( "{0} ", item );
34
35         // sort theList in ascending order (the default)
36         var ascending =
37             from letter in theList
38             orderby letter
39             select letter;
40
41         // display ascending order
42         Console.WriteLine( "\nAscending order:" );
43         foreach ( var item in ascending )
44             Console.Write( "{0} ", item );
45
46         // sort the list in descending order
47         var descending =
48             from letter in theList
49             orderby letter descending
50             select letter;
51

```

```
52 // display descending order
53 Console.WriteLine( "\nDescending order:" );
54 foreach ( var item in descending )
55     Console.Write( "{0} ", item );
56
57 // display distinct letters in the List in ascending order
58 Console.WriteLine( "\nAscending order, no duplicates:" );
59 foreach ( var item in ascending.Distinct() )
60     Console.Write( "{0} ", item );
61
62     Console.WriteLine();
63 } // end Main
64 } // end class LINQLetterList
```

Generated List:
v l k k y e e b b e i p v e m s k m g c r v i u d o o q z e
Ascending order:
b b c d e e e e e g i i k k k l m m o o p q r s u v v v y z
Descending order:
z y v v v u s r q p o o m m l k k k i i g e e e e e d c b b
Ascending order, no duplicates:
b c d e g i k l m o p q r s u v y z

10

Classes and Objects: A Deeper Look: Solutions

*But what, to serve
our private ends,
Forbids the cheating
of our friends?*

—Charles Churchill

*This above all: to thine own self
be true.*

—William Shakespeare.

Objectives

In this chapter you'll learn:

- Encapsulation and data hiding.
- To use keyword `this`.
- To use `static` variables and methods.
- To use `readonly` fields.
- To take advantage of C#'s memory-management features.
- To use the IDEs **Class View** and **Object Browser** windows.
- To use object initializers to create an object and initialize it in the same statement.



Self-Review Exercises

10.1 Fill in the blanks in each of the following statements:

- a) `string` class static method _____ is similar to method `Console.WriteLine`, but returns a formatted string rather than displaying a string in a console window.

ANS: `Format`.

- b) If a method contains a local variable with the same name as one of its class's fields, the local variable _____ the field in that method's scope.

ANS: hides.

- c) The _____ is called by the garbage collector just before it reclaims an object's memory.

ANS: destructor.

- d) If a class declares constructors, the compiler will not create a(n) _____.

ANS: default constructor

- e) An object's _____ method can be called implicitly when an object appears in code where a string is needed.

ANS: `ToString`.

- f) Composition is sometimes referred to as a(n) _____ relationship.

ANS: *has-a*.

- g) A(n) _____ variable represents classwide information that is shared by all the objects of the class.

ANS: `static`.

- h) The _____ states that code should be granted only the amount of access needed to accomplish its designated task.

ANS: principle of least privilege.

- i) Declaring an instance variable with keyword `or` _____ specifies that the variable is not modifiable.

ANS: `readonly`.

- j) A(n) _____ consists of a data representation and the operations that can be performed on the data.

ANS: abstract data type (ADT).

- k) The public methods of a class are also known as the class's _____ or _____.

ANS: public services, public interface.

10.2 Suppose class `Book` defines properties `Title`, `Author` and `Year`. Use an object initializer to create an object of class `Book` and initialize its properties.

ANS: `new Book { Title = "Visual C# 2008 HTP", _
Author = "Deitel", Year = 2010 }`

Exercises

10.3 (*Rectangle Class*) Create class `Rectangle`. The class has attributes `length` and `width`, each of which defaults to 1. It has read-only properties that calculate the `Perimeter` and the `Area` of the rectangle. It has properties for both `length` and `width`. The set accessors should verify that `length` and `width` are each floating-point numbers greater than 0.0 and less than 20.0. Write an application to test class `Rectangle`.

ANS:

```
1 // Exercise 10.3 Solution: Rectangle.cs
2 // Definition of class Rectangle
3 using System;
```

```
4
5 public class Rectangle
6 {
7     private double length; // the length of the rectangle
8     private double width; // the width of the rectangle
9
10    // constructor without parameters
11    public Rectangle()
12    {
13        Length = 1.0;
14        Width = 1.0;
15    } // end Rectangle parameterless constructor
16
17    // constructor with length and width supplied
18    public Rectangle( double theLength, double theWidth )
19    {
20        Length = theLength;
21        Width = theWidth;
22    } // end Rectangle two-parameter constructor
23
24    // property that gets and sets the length
25    public double Length
26    {
27        get
28        {
29            return length;
30        } // end get
31        set
32        {
33            if ( value > 0.0 && value < 20.0 )
34                length = value;
35            else
36                throw new ArgumentOutOfRangeException( "length", value,
37                    "length must be greater than 0 and less than 20" );
38        } // end set
39    } // end property Length
40
41    // property that gets and sets the width
42    public double Width
43    {
44        get
45        {
46            return width;
47        } // end get
48        set
49        {
50            if ( value > 0.0 && value < 20.0 )
51                width = value;
52            else
53                throw new ArgumentOutOfRangeException( "width", value,
54                    "width must be greater than 0 and less than 20" );
55        } // end set
56    } // end property Width
57
```

```

58 // read-only property that calculates the perimeter
59 public double Perimeter
60 {
61     get
62     {
63         return 2 * Length + 2 * Width;
64     } // end get
65 } // end property Perimeter
66
67 // read-only property that calculates the area
68 public double Area
69 {
70     get
71     {
72         return Length * Width;
73     } // end get
74 } // end property Area
75
76 // convert to string
77 public override string ToString()
78 {
79     return string.Format( "{0}: {1}\n{2}: {3}\n{4}: {5}\n{6}: {7}",
80         "Length", Length, "Width", Width,
81         "Perimeter", Perimeter, "Area", Area );
82 } // end method ToString
83 } // end class Rectangle

```

```

1 // Exercise 10.3 Solution: RectangleTest.cs
2 // Application tests class Rectangle.
3 using System;
4
5 public class RectangleTest
6 {
7     public static void Main( string[] args )
8     {
9         Rectangle rectangle = new Rectangle();
10
11         int choice = GetMenuChoice();
12
13         while ( choice != 3 )
14         {
15             try
16             {
17                 switch ( choice )
18                 {
19                     case 1:
20                         Console.Write( "Enter length: " );
21                         rectangle.Length = Convert.ToDouble( Console.ReadLine() );
22                         break;
23                     case 2:
24                         Console.Write( "Enter width: " );
25                         rectangle.Width = Convert.ToDouble( Console.ReadLine() );
26                         break;
27                 } // end switch

```

```

28
29         Console.WriteLine( rectangle.ToString() );
30     } // end try
31     catch ( ArgumentOutOfRangeException ex )
32     {
33         Console.WriteLine( ex.Message );
34     } // end catch
35
36     Console.WriteLine();
37
38     choice = GetMenuChoice();
39 } // end while
40 } // end Main
41
42 // displays a menu and returns the chosen value
43 private static int GetMenuChoice()
44 {
45     Console.WriteLine( "1. Set Length" );
46     Console.WriteLine( "2. Set Width" );
47     Console.WriteLine( "3. Exit" );
48     Console.Write( "Choice: " );
49
50     return Convert.ToInt32( Console.ReadLine() );
51 } // end method GetMenuChoice
52 } // end class RectangleTest

```

```

1. Set Length
2. Set Width
3. Exit
Choice: 1
Enter length: 10
Length: 10
Width: 1
Perimeter: 22
Area: 10

```

```

1. Set Length
2. Set Width
3. Exit
Choice: 2
Enter width: 15
Length: 10
Width: 15
Perimeter: 50
Area: 150

```

```

1. Set Length
2. Set Width
3. Exit
Choice: 2
Enter width: 5
Length: 10
Width: 5
Perimeter: 30
Area: 50

```

```

1. Set Length
2. Set Width
3. Exit
Choice: 3

```

10.4 (*Modifying the Internal Data Representation of a Class*) It would be perfectly reasonable for the `Time2` class of Fig. 10.5 to represent the time internally as the number of seconds since midnight rather than the three integer values `hour`, `minute` and `second`. Clients could use the same public methods and properties to get the same results. Modify the `Time2` class of Fig. 10.5 to implement the `Time2` as the number of seconds since midnight and show that no change is visible to the clients of the class by using the same test application from Fig. 10.6.

ANS:

```

1  // Exercise 10.4 Solution: Time2.cs
2  // Time2 class definition maintains the time in 24-hour format.
3  using System; // for class ArgumentOutOfRangeException
4
5  public class Time2
6  {
7      private int totalSeconds;
8
9      // Time2 constructor: hour, minute and second supplied
10     public Time2( int h = 0, int m = 0, int s = 0 )
11     {
12         SetTime( h, m, s ); // invoke SetTime to validate time
13     } // end Time2 three-argument constructor
14
15     // Time2 constructor: another Time2 object supplied as an argument
16     public Time2( Time2 time )
17         : this( time.Hour, time.Minute, time.Second ) { }
18
19     // set a new time value using universal time; ensure that
20     // the data remains consistent by setting invalid values to zero
21     public void SetTime( int h, int m, int s )
22     {
23         Hour = h; // set the Hour property
24         Minute = m; // set the Minute property
25         Second = s; // set the Second property
26     } // end method SetTime
27
28     // property that gets and sets the hour
29     public int Hour
30     {
31         get
32         {
33             return ( totalSeconds / 3600 );
34         } // end get
35         set
36         {
37             if ( value >= 0 && value < 24 )
38                 totalSeconds = ( value * 3600 ) + ( Minute * 60 ) + Second;

```

```

39         else
40             throw new ArgumentOutOfRangeException(
41                 "hour", value, "hour must be 0-23" );
42     } // end set
43 } // end property Hour
44
45 // property that gets and sets the minute
46 public int Minute
47 {
48     get
49     {
50         return ( ( totalSeconds % 3600 ) / 60 );
51     } // end get
52     set
53     {
54         if ( value >= 0 && value < 60 )
55             totalSeconds = ( Hour * 3600 ) + ( value * 60 ) + Second;
56         else
57             throw new ArgumentOutOfRangeException(
58                 "minute", value, "minute must be 0-59" );
59     } // end set
60 } // end property Minute
61
62 // property that gets and sets the second
63 public int Second
64 {
65     get
66     {
67         return ( totalSeconds % 60 );
68     } // end get
69     set
70     {
71         if ( value >= 0 && value < 60 )
72             totalSeconds = ( Hour * 3600 ) + ( Minute * 60 ) + value;
73         else
74             throw new ArgumentOutOfRangeException(
75                 "second", value, "second must be 0-59" );
76     } // end set
77 } // end property Second
78
79 // convert to string in universal-time format (HH:MM:SS)
80 public string ToUniversalString()
81 {
82     return string.Format(
83         "{0:D2}:{1:D2}:{2:D2}", Hour, Minute, Second );
84 } // end method ToUniversalString
85
86 // convert to string in standard-time format (H:MM:SS AM or PM)
87 public override string ToString()
88 {
89     return string.Format( "{0}:{1:D2}:{2:D2} {3}",
90         ( ( Hour == 0 || Hour == 12 ) ? 12 : Hour % 12 ),
91         Minute, Second, ( Hour < 12 ? "AM" : "PM" ) );
92 } // end method ToString
93 } // end class Time2

```

```

1 // Exercise 10.4 Solution: Time2Test.cs
2 // Overloaded constructors used to initialize Time2 objects.
3 using System;
4
5 public class Time2Test
6 {
7     public static void Main( string[] args )
8     {
9         Time2 t1 = new Time2(); // 00:00:00
10        Time2 t2 = new Time2( 2 ); // 02:00:00
11        Time2 t3 = new Time2( 21, 34 ); // 21:34:00
12        Time2 t4 = new Time2( 12, 25, 42 ); // 12:25:42
13        Time2 t5 = new Time2( t4 ); // 12:25:42
14        Time2 t6; // initialized later in the program
15
16        Console.WriteLine( "Constructed with:\n" );
17        Console.WriteLine( "t1: all arguments defaulted" );
18        Console.WriteLine( "    {0}", t1.ToUniversalString() ); // 00:00:00
19        Console.WriteLine( "    {0}\n", t1.ToString() ); // 12:00:00 AM
20
21        Console.WriteLine(
22            "t2: hour specified; minute and second defaulted" );
23        Console.WriteLine( "    {0}", t2.ToUniversalString() ); // 02:00:00
24        Console.WriteLine( "    {0}\n", t2.ToString() ); // 2:00:00 AM
25
26        Console.WriteLine(
27            "t3: hour and minute specified; second defaulted" );
28        Console.WriteLine( "    {0}", t3.ToUniversalString() ); // 21:34:00
29        Console.WriteLine( "    {0}\n", t3.ToString() ); // 9:34:00 PM
30
31        Console.WriteLine( "t4: hour, minute and second specified" );
32        Console.WriteLine( "    {0}", t4.ToUniversalString() ); // 12:25:42
33        Console.WriteLine( "    {0}\n", t4.ToString() ); // 12:25:42 PM
34
35        Console.WriteLine( "t5: Time2 object t4 specified" );
36        Console.WriteLine( "    {0}", t5.ToUniversalString() ); // 12:25:42
37        Console.WriteLine( "    {0}\n", t5.ToString() ); // 12:25:42 PM
38
39        // attempt to initialize t6 with invalid values
40        try
41        {
42            t6 = new Time2( 27, 74, 99 ); // invalid values
43        } // end try
44        catch ( ArgumentOutOfRangeException ex )
45        {
46            Console.WriteLine( "\nException while initializing t6:" );
47            Console.WriteLine( ex.Message );
48        } // end catch
49    } // end Main
50 } // end class Time2Test

```

Constructed with:

t1: all arguments defaulted
00:00:00
12:00:00 AM

t2: hour specified; minute and second defaulted
02:00:00
2:00:00 AM

t3: hour and minute specified; second defaulted
21:34:00
9:34:00 PM

t4: hour, minute and second specified
12:25:42
12:25:42 PM

t5: Time2 object t4 specified
12:25:42
12:25:42 PM

Exception while initializing t6:
hour must be 0-23
Parameter name: hour
Actual value was 27.

10.5 (*Savings-Account Class*) Create the class `SavingsAccount`. Use the static variable `annualInterestRate` to store the annual interest rate for all account holders. Each object of the class contains a private instance variable `savingsBalance`, indicating the amount the saver currently has on deposit. Provide method `CalculateMonthlyInterest` to calculate the monthly interest by multiplying the `savingsBalance` by `annualInterestRate` divided by 12—this interest should be added to `savingsBalance`. Provide static method `ModifyInterestRate` to set the `annualInterestRate` to a new value. Write an application to test class `SavingsAccount`. Create two `savingsAccount` objects, `saver1` and `saver2`, with balances of \$2000.00 and \$3000.00, respectively. Set `annualInterestRate` to 4%, then calculate the monthly interest and display the new balances for both savers. Then set the `annualInterestRate` to 5%, calculate the next month's interest and display the new balances for both savers.

ANS:

```

1 // Exercise 10.5 Solution: SavingAccount
2 // SavingAccount class definition
3 public class SavingAccount
4 {
5     // interest rate for all accounts
6     private static decimal annualInterestRate = 0M;
7
8     private decimal savingsBalance; // balance for current account
9
10    // constructor, creates a new account with the specified balance
11    public SavingAccount( decimal balance )
12    {

```

```

13     savingsBalance = balance;
14 } // end constructor
15
16 // add the monthly interest to the balance
17 public void CalculateMonthlyInterest()
18 {
19     savingsBalance += savingsBalance * ( annualInterestRate / 12M );
20 } // end method CalculateMonthlyInterest
21
22 // modify interest rate
23 public static void ModifyInterestRate( decimal newRate )
24 {
25     if ( newRate >= 0M && newRate <= 1M )
26         annualInterestRate = newRate;
27     else
28         throw new ArgumentOutOfRangeException( "annualInterestRate",
29             newRate, "interest rate must be in the range 0 to 1" );
30 } // end method ModifyInterestRate
31
32 // get string representation of SavingAccount
33 public override string ToString()
34 {
35     return string.Format( "{0:C}", savingsBalance );
36 } // end method ToString
37 } // end class SavingAccount

```

```

1 // Exercise 10.5 Solution: SavingAccountTest.cs
2 // Application that tests SavingAccount class
3 using System;
4
5 public class SavingAccountTest
6 {
7     public static void Main( string[] args )
8     {
9         SavingAccount saver1 = new SavingAccount( 2000M );
10        SavingAccount saver2 = new SavingAccount( 3000M );
11        SavingAccount.ModifyInterestRate( 0.04M );
12
13        Console.WriteLine( "Monthly balances for one year at .04" );
14        Console.WriteLine( "Balances:" );
15
16        Console.WriteLine( "{0,10}{1,10}{2,10}", string.Empty,
17            "Saver 1", "Saver 2" );
18        Console.WriteLine( "{0,-10}{1,10}{2,10}", "Base",
19            saver1.ToString(), saver2.ToString() );
20
21        for ( int month = 1; month <= 12; month++ )
22        {
23            string monthLabel = string.Format( "Month {0}:", month );
24            saver1.CalculateMonthlyInterest();
25            saver2.CalculateMonthlyInterest();
26

```

```

27         Console.WriteLine( "{0,-10}{1,10}{2,10}", monthLabel,
28             saver1.ToString(), saver2.ToString() );
29     } // end for
30
31     SavingAccount.ModifyInterestRate( .05M );
32     saver1.CalculateMonthlyInterest();
33     saver2.CalculateMonthlyInterest();
34
35     Console.WriteLine( "\nAfter setting interest rate to .05" );
36     Console.WriteLine( "Balances:" );
37     Console.WriteLine( "{0,10}{1,10}{2,10}", string.Empty,
38         "Saver 1", "Saver 2" );
39     Console.WriteLine( "{0,-10}{1,10}{2,10}", "Month 13:",
40         saver1.ToString(), saver2.ToString() );
41 } // end Main
42 } // end class SavingAccountTest

```

Monthly balances for one year at .04
Balances:

	Saver 1	Saver 2
Base	\$2,000.00	\$3,000.00
Month 1:	\$2,006.67	\$3,010.00
Month 2:	\$2,013.36	\$3,020.03
Month 3:	\$2,020.07	\$3,030.10
Month 4:	\$2,026.80	\$3,040.20
Month 5:	\$2,033.56	\$3,050.33
Month 6:	\$2,040.33	\$3,060.50
Month 7:	\$2,047.14	\$3,070.70
Month 8:	\$2,053.96	\$3,080.94
Month 9:	\$2,060.81	\$3,091.21
Month 10:	\$2,067.68	\$3,101.51
Month 11:	\$2,074.57	\$3,111.85
Month 12:	\$2,081.48	\$3,122.22

After setting interest rate to .05
Balances:

	Saver 1	Saver 2
Month 13:	\$2,090.16	\$3,135.23

10.6 (*Enhancing Class Date*) Modify class Date of Fig. 10.7 to perform error checking on the initializer values for instance variables month, day and year (class Date currently validates only the month and day). You'll need to convert the auto-implemented property Year into instance variable year with an associated Year property. Provide method NextDay to increment the day by 1. The Date object should always maintain valid data and throw exceptions when attempts are made to set invalid values. Write an application that tests the NextDay method in a loop that displays the date during each iteration of the loop to illustrate that the NextDay method works correctly. Test the following cases:

- incrementing to the next month and
- incrementing to the next year.

ANS:

```

1  // Exercise 10.6 Solution: Date.cs
2  // Date class declaration.
3  using System;
4
5  public class Date
6  {
7      private int month; // 1-12
8      private int day; // 1-31 based on month
9      private int year; // >0
10
11     private static readonly int[] DAYSPERMONTH =
12         { 0, 31, 28, 31, 30, 31, 30, 31, 31, 30, 31, 30, 31 };
13
14     // constructor: use property Month to confirm proper value for month;
15     // use property Day to confirm proper value for day
16     public Date( int theMonth, int theDay, int theYear )
17     {
18         Month = theMonth; // validate month
19         Year = theYear; // validate year
20         Day = theDay; // validate day
21         Console.WriteLine( "Date object constructor for date {0}", this );
22     } // end Date constructor
23
24     // property that gets and sets the year
25     public int Year
26     {
27         get
28         {
29             return year;
30         } // end get
31         private set
32         {
33             year = CheckYear( value );
34         } // end set
35     } // end property year
36
37     // property that gets and sets the month
38     public int Month
39     {
40         get
41         {
42             return month;
43         } // end get
44         private set // make writing inaccessible outside the class
45         {
46             month = CheckMonth( value );
47         } // end set
48     } // end property Month
49
50     // property that gets and sets the day
51     public int Day
52     {

```

```

53     get
54     {
55         return day;
56     } // end get
57     private set // make writing inaccessible outside the class
58     {
59         day = CheckDay( value );
60     } // end set
61 } // end property Day
62
63 // increment the day and check if doing so will change the month
64 public void NextDay()
65 {
66     if ( !endOfMonth() )
67         ++Day;
68     else
69     {
70         Day = 1;
71         NextMonth();
72     }
73 } // end method NextDay
74
75 // increment the month and check if doing so will change the year
76 public void NextMonth()
77 {
78     if ( Month < 12 )
79         ++Month;
80     else
81     {
82         Month = 1;
83         ++Year;
84     }
85 } // end method NextMonth
86
87 // return a string of the form month/day/year
88 public override string ToString()
89 {
90     return string.Format( "{0}/{1}/{2}", Month, Day, Year );
91 } // end method ToString
92
93 // utility method to confirm proper year value
94 private int CheckYear( int testYear )
95 {
96     if ( testYear > 0 ) // validate year
97         return testYear;
98     else // year is invalid
99         throw new ArgumentOutOfRangeException(
100             "year", testYear, "year must greater than 0" );
101 } // end method CheckYear
102
103 // utility method to confirm proper month value
104 private int CheckMonth( int testMonth )
105 {
106     if ( testMonth > 0 && testMonth <= 12 ) // validate month
107         return testMonth;

```

```

108     else // month is invalid
109         throw new ArgumentOutOfRangeException(
110             "month", testMonth, "month must be 1-12" );
111 } // end method CheckMonth
112
113 // utility method to confirm proper day value based on month and year
114 private int CheckDay( int testDay )
115 {
116     // Check if day in range for month
117     if ( testDay > 0 && testDay <= DAYSPERMONTH[ Month ] )
118         return testDay;
119
120     // Check for leap year
121     if ( testDay == 29 && leapYear() )
122         return testDay;
123
124     throw new ArgumentOutOfRangeException(
125         "day", testDay, "day out of range for current month/year" );
126 } // end method CheckDay
127
128 // check for end of month
129 private bool endOfMonth()
130 {
131     if ( leapYear() && Month == 2 && Day == 29 )
132         return true;
133     else
134         return Day == DAYSPERMONTH[ Month ];
135 } // end method endOfMonth
136
137 private bool leapYear()
138 {
139     return Month == 2 && ( Year % 400 == 0 ||
140         ( Year % 4 == 0 && Year % 100 != 0 ) );
141 } // end method leapYear
142 } // end class Date

```

```

1 // Exercise 10.6 Solution: DateTest
2 // Application tests Date class with year validation,
3 // NextDay and NextMonth methods.
4 using System;
5
6 public class DateTest
7 {
8     // method Main begins execution of C# application
9     public static void Main( string[] args )
10    {
11        Console.WriteLine( "Checking increment" );
12        Date testDate = new Date( 12, 8, 1980 );
13
14        // test incrementing of day, month and year
15        for ( int counter = 0; counter < 40; counter++ )
16        {

```

```

17         testDate.NextDay();
18         Console.WriteLine( "Incremented Date: {0}",
19             testDate.ToString() );
20     } // end for
21 } // end Main
22 } // end class DateTest

```

```

Checking increment
Date object constructor for date 12/8/1980
Incremented Date: 12/9/1980
Incremented Date: 12/10/1980
Incremented Date: 12/11/1980
Incremented Date: 12/12/1980
Incremented Date: 12/13/1980
Incremented Date: 12/14/1980
Incremented Date: 12/15/1980
Incremented Date: 12/16/1980
Incremented Date: 12/17/1980
Incremented Date: 12/18/1980
Incremented Date: 12/19/1980
Incremented Date: 12/20/1980
Incremented Date: 12/21/1980
Incremented Date: 12/22/1980
Incremented Date: 12/23/1980
Incremented Date: 12/24/1980
Incremented Date: 12/25/1980
Incremented Date: 12/26/1980
Incremented Date: 12/27/1980
Incremented Date: 12/28/1980
Incremented Date: 12/29/1980
Incremented Date: 12/30/1980
Incremented Date: 12/31/1980
Incremented Date: 1/1/1981
Incremented Date: 1/2/1981
Incremented Date: 1/3/1981
Incremented Date: 1/4/1981
Incremented Date: 1/5/1981
Incremented Date: 1/6/1981
Incremented Date: 1/7/1981
Incremented Date: 1/8/1981
Incremented Date: 1/9/1981
Incremented Date: 1/10/1981
Incremented Date: 1/11/1981
Incremented Date: 1/12/1981
Incremented Date: 1/13/1981
Incremented Date: 1/14/1981
Incremented Date: 1/15/1981
Incremented Date: 1/16/1981
Incremented Date: 1/17/1981

```

10.7 (Complex Numbers) Create a class called `Complex` for performing arithmetic with complex numbers. Complex numbers have the form

$$\text{realPart} + \text{imaginaryPart} * i$$

where i is

$$\sqrt{-1}$$

Write an application to test your class. Use floating-point variables to represent the private data of the class. Provide a constructor that enables an object of this class to be initialized when it's declared. Provide a parameterless constructor with default values in case no initializers are provided. Provide public methods that perform the following operations:

- a) Add two Complex numbers: The real parts are added together and the imaginary parts are added together.
- b) Subtract two Complex numbers: The real part of the right operand is subtracted from the real part of the left operand, and the imaginary part of the right operand is subtracted from the imaginary part of the left operand.
- c) Return a string representation of a Complex number in the form (a, b), where a is the real part and b is the imaginary part.

ANS:

```

1  // Exercise 10.7 Solution: Complex.cs
2  // Definition of class Complex
3  public class Complex
4  {
5      private double real;
6      private double imaginary;
7
8      // Initialize both parts to 0
9      public Complex() : this( 0.0, 0.0 ) { }
10
11     // Initialize real part to r and imaginary part to i
12     public Complex( double r, double i )
13     {
14         real = r;
15         imaginary = i;
16     } // end Complex two-parameter constructor
17
18     // Add two Complex numbers
19     public Complex Add( Complex right )
20     {
21         return new Complex( real + right.real,
22                             imaginary + right.imaginary );
23     } // end method Add
24
25     // Subtract two Complex numbers
26     public Complex Subtract( Complex right )
27     {
28         return new Complex( real - right.real,
29                             imaginary - right.imaginary );
30     } // end method Subtract
31
32     // Return string representation of a Complex number
33     public override string ToString()
34     {
35         return string.Format( "( {0:F1}, {1:F1} )", real, imaginary );
36     } // end method ToString;
37 } // end class Complex

```

```

1 // Exercise 10.7: ComplexTest.cs
2 // Test the Complex number class
3 using System;
4
5 public class ComplexTest
6 {
7     public static void Main( string[] args )
8     {
9         // initialize two numbers
10        Complex a = new Complex( 9.5, 7.7 );
11        Complex b = new Complex( 1.2, 3.1 );
12
13        Console.WriteLine( "a = {0}", a.ToString() );
14        Console.WriteLine( "b = {0}", b.ToString() );
15        Console.WriteLine( "a + b = {0}", a.Add( b ).ToString() );
16        Console.WriteLine( "a - b = {0}", a.Subtract( b ).ToString() );
17    } // end Main
18 } // end class ComplexTest

```

```

a = ( 9.5, 7.7 )
b = ( 1.2, 3.1 )
a + b = ( 10.7, 10.8 )
a - b = ( 8.3, 4.6 )

```

10.8 (*Set of Integers*) Create class `IntegerSet`. Each `IntegerSet` object can hold integers in the range 0–100. The set is represented by an array of `bool`s. Array element `a[i]` is `true` if integer `i` is in the set. Array element `a[j]` is `false` if integer `j` is not in the set. The parameterless constructor initializes the array to the “empty set” (i.e., a set whose array representation contains all `false` values).

Provide the following methods:

- Method `Union` creates a third set that is the set-theoretic union of two existing sets (i.e., an element of the third set’s array is set to `true` if that element is `true` in either or both of the existing sets—otherwise, the element of the third set is set to `false`).
- Method `Intersection` creates a third set which is the set-theoretic intersection of two existing sets (i.e., an element of the third set’s array is set to `false` if that element is `false` in either or both of the existing sets—otherwise, the element of the third set is set to `true`).
- Method `InsertElement` inserts a new integer k into a set (by setting `a[k]` to `true`).
- Method `DeleteElement` deletes integer m (by setting `a[m]` to `false`).
- Method `ToString` returns a string containing a set as a list of numbers separated by spaces. Include only those elements that are present in the set. Use `---` to represent an empty set.
- Method `IsEqualTo` determines whether two sets are equal.

Write an application to test class `IntegerSet`. Instantiate several `IntegerSet` objects. Test that all your methods work properly.

ANS:

```

1 // Exercise 10.8 Solution: IntegerSet.cs
2 // IntegerSet class definition
3 public class IntegerSet
4 {
5     private const int SETSIZE = 101;
6     private bool[] set;

```

```
7
8 // parameterless constructor, creates an empty set
9 public IntegerSet()
10 {
11     set = new bool[ SETSIZE ];
12 } // end parameterless constructor
13
14 // constructor creates a set from array of integers
15 public IntegerSet( int[] array )
16     : this()
17 {
18     for ( int i = 0; i < array.Length; i++ )
19         InsertElement( array[ i ] );
20 } // end constructor
21
22 // return string representation of set
23 public override string ToString()
24 {
25     string setString = "{ ";
26
27     // get set elements
28     for ( int count = 0; count < SETSIZE; count++ )
29     {
30         if ( set[ count ] )
31         {
32             setString += count + " ";
33             empty = false; // set is not empty
34         } // end if
35     } // end for
36
37     // empty set
38     if ( setString == "{ " )
39         setString += "--- "; // display an empty set
40
41     setString += "}";
42     return setString;
43 } // end method ToString
44
45 // returns the union of two sets
46 public IntegerSet Union( IntegerSet integerSet )
47 {
48     IntegerSet temp = new IntegerSet();
49
50     for ( int count = 0; count < SETSIZE; count++ )
51         temp.set[ count ] = ( set[ count ] || integerSet.set[ count ] );
52
53     return temp;
54 } // end method Union
55
56 // returns the intersection of two sets
57 public IntegerSet Intersection( IntegerSet integerSet )
58 {
59     IntegerSet temp = new IntegerSet();
60
```

```

61     for ( int count = 0; count < SETSIZE; count++ )
62         temp.set[ count ] =
63             ( set[ count ] && integerSet.set[ count ] );
64
65     return temp;
66 } // end method Intersection
67
68 // insert a new integer into this set
69 public void InsertElement( int insertInteger )
70 {
71     if ( ValidEntry( insertInteger ) )
72         set[ insertInteger ] = true;
73 } // end method InsertElement
74
75 // remove an integer from this set
76 public void DeleteElement( int deleteInteger )
77 {
78     if ( ValidEntry( deleteInteger ) )
79         set[ deleteInteger ] = false;
80 } // end method DeleteElement
81
82 // determine if two sets are equal
83 public bool IsEqualTo( IntegerSet integerSet )
84 {
85     for ( int count = 0; count < SETSIZE; count++ )
86     {
87         if ( set[ count ] != integerSet.set[ count ] )
88             return false; // sets are not-equal
89     } // end for
90
91     return true; // sets are equal
92 } // end method IsEqualTo
93
94 // determine if input is valid
95 public bool ValidEntry( int input )
96 {
97     return input >= 0 && input < SETSIZE;
98 } // end method ValidEntry
99 } // end class IntegerSet

```

```

1 // Exercise 10.8 Solution: IntegerSetTest.cs
2 // Application that tests IntegerSet
3 using System;
4
5 public class IntegerSetTest
6 {
7     public static void Main( string[] args )
8     {
9         // initialize two sets
10        Console.WriteLine( "Input Set A" );
11        IntegerSet set1 = InputSet();
12        Console.WriteLine( "\nInput Set B" );
13        IntegerSet set2 = InputSet();

```

```

14
15     IntegerSet union = set1.Union( set2 );
16     IntegerSet intersection = set1.Intersection( set2 );
17
18     // prepare output
19     Console.WriteLine( "\nSet A contains elements:" );
20     Console.WriteLine( set1.ToString() );
21     Console.WriteLine( "\nSet B contains elements:" );
22     Console.WriteLine( set2.ToString() );
23     Console.WriteLine(
24         "\nUnion of Set A and Set B contains elements:" );
25     Console.WriteLine( union.ToString() );
26     Console.WriteLine(
27         "\nIntersection of Set A and Set B contains elements:" );
28     Console.WriteLine( intersection.ToString() );
29
30     // test whether two sets are equal
31     if ( set1.IsEqualTo( set2 ) )
32         Console.WriteLine( "\nSet A is equal to set B" );
33     else
34         Console.WriteLine( "\nSet A is not equal to set B" );
35
36     // test insert and delete
37     Console.WriteLine( "\nInserting 77 into set A..." );
38     set1.InsertElement( 77 );
39     Console.WriteLine( "\nSet A now contains elements:" );
40     Console.WriteLine( set1.ToString() );
41
42     Console.WriteLine( "\nDeleting 77 from set A..." );
43     set1.DeleteElement( 77 );
44     Console.WriteLine( "\nSet A now contains elements:" );
45     Console.WriteLine( set1.ToString() );
46
47     // test constructor
48     int[] intArray = { 25, 67, 2, 9, 99, 105, 45, -5, 100, 1 };
49     IntegerSet set3 = new IntegerSet( intArray );
50
51     Console.WriteLine( "\nNew Set contains elements:" );
52     Console.WriteLine( set3.ToString() );
53 } // end Main
54
55 // creates a new set by reading numbers from the user
56 private static IntegerSet InputSet()
57 {
58     IntegerSet temp = new IntegerSet();
59
60     Console.Write( "Enter number (-1 to end): " );
61     int number = Convert.ToInt32( Console.ReadLine() );
62
63     while ( number != -1 )
64     {
65         temp.InsertElement( number );
66
67         Console.Write( "Enter number (-1 to end): " );

```

```

68         number = Convert.ToInt32( Console.ReadLine() );
69     } // end while
70
71     return temp;
72 } // end method InputSet
73 } // end class IntegerSetTest

```

```

Input Set A
Enter number (-1 to end): 6
Enter number (-1 to end): 2
Enter number (-1 to end): 49
Enter number (-1 to end): 42
Enter number (-1 to end): 98
Enter number (-1 to end): -1

Input Set B
Enter number (-1 to end): 42
Enter number (-1 to end): 36
Enter number (-1 to end): 75
Enter number (-1 to end): 4
Enter number (-1 to end): 3
Enter number (-1 to end): 2
Enter number (-1 to end): 1
Enter number (-1 to end): -1

Set A contains elements:
{ 2 6 42 49 98 }

Set B contains elements:
{ 1 2 3 4 36 42 75 }

Union of Set A and Set B contains elements:
{ 1 2 3 4 6 36 42 49 75 98 }

Intersection of Set A and Set B contains elements:
{ 2 42 }

Set A is not equal to set B

Inserting 77 into set A...

Set A now contains elements:
{ 2 6 42 49 77 98 }

Deleting 77 from set A...

Set A now contains elements:
{ 2 6 42 49 98 }

New Set contains elements:
{ 1 2 9 25 45 67 99 100 }

```

10.9 (*Rational Numbers*) Create a class called `Rational` for performing arithmetic with fractions. Write an application to test your class. Use integer variables to represent the private instance variables of the class—the numerator and the denominator. Provide a constructor that enables an object

of this class to be initialized when it's declared. The constructor should store the fraction in reduced form. The fraction

$2/4$

is equivalent to $1/2$ and would be stored in the object as 1 in the numerator and 2 in the denominator. Provide a parameterless constructor with default values in case no initializers are provided. Provide public methods that perform each of the following operations (all calculation results should be stored in a reduced form):

- a) Add two Rational numbers.
- b) Subtract two Rational numbers.
- c) Multiply two Rational numbers.
- d) Divide two Rational numbers.
- e) Display Rational numbers in the form a/b, where a is the numerator and b is the denominator.
- f) Display Rational numbers in floating-point format. (Consider providing formatting capabilities that enable the user of the class to specify the number of digits of precision to the right of the decimal point.)

ANS:

```

1 // Exercise 10.9 Solution: Rational.cs
2 // Rational class declaration.
3 public class Rational
4 {
5     private int numerator; // numerator of the fraction
6     private int denominator; // denominator of the fraction
7
8     // parameterless constructor, initializes this Rational to 1
9     public Rational() : this( 1, 1 ) { }
10
11     // initialize numerator and denominator
12     public Rational( int theNumerator, int theDenominator )
13     {
14         numerator = theNumerator;
15         denominator = theDenominator;
16         Reduce();
17     } // end two-parameter constructor
18
19     // add two Rational numbers
20     public Rational Sum( Rational right )
21     {
22         int resultDenominator = denominator * right.denominator;
23         int resultNumerator = numerator * right.denominator +
24             right.numerator * denominator;
25
26         return new Rational( resultNumerator, resultDenominator );
27     } // end method Sum
28
29     // subtract two Rational numbers
30     public Rational Subtract( Rational right )
31     {
32         int resultDenominator = denominator * right.denominator;
33         int resultNumerator = numerator * right.denominator -
34             right.numerator * denominator;

```

```
35
36     return new Rational( resultNumerator, resultDenominator );
37 } // end method Subtract
38
39 // multiply two Rational numbers
40 public Rational Multiply( Rational right )
41 {
42     return new Rational( numerator * right.numerator,
43                          denominator * right.denominator );
44 } // end method Multiply
45
46 // divide two Rational numbers
47 public Rational Divide( Rational right )
48 {
49     return new Rational( numerator * right.denominator,
50                          denominator * right.numerator );
51 } // end method Divide
52
53 // reduce the fraction
54 private void Reduce()
55 {
56     int gcd = 0;
57     int smaller;
58
59     // find the greatest common denominator of the two numbers
60     if ( numerator < denominator )
61         smaller = numerator;
62     else
63         smaller = denominator;
64
65     for ( int divisor = smaller; divisor >= 2; divisor-- )
66     {
67         if ( numerator % divisor == 0 && denominator % divisor == 0 )
68         {
69             gcd = divisor;
70             break;
71         } // end if
72     } // end for
73
74     // divide both the numerator and denominator by the gcd
75     if ( gcd != 0 )
76     {
77         numerator /= gcd;
78         denominator /= gcd;
79     } // end if
80 } // end method Reduce
81
82 // return string representation of a Rational number
83 public override string ToString()
84 {
85     return numerator + "/" + denominator;
86 } // end method ToString
87
```

```

88 // return floating-point string representation of
89 // a Rational number
90 public string ToFloatString( int digits )
91 {
92     double value = ( double ) numerator / denominator;
93     // builds a formatting string that specifies the precision
94     // based on the digits parameter
95     return string.Format( "{0:F" + digits + "}", value );
96 } // end method ToFloatString
97 } // end class Rational

```

```

1 // Exercise 10.9 Solution: RationalTest.cs
2 // Application tests class Rational.
3 using System;
4
5 public class RationalTest
6 {
7     public static void Main( string[] args )
8     {
9         int numerator; // the numerator of a fraction
10        int denominator; // the denominator of a fraction
11        int digits; // digits to display in floating point format
12        Rational rational1; // the first rational number
13        Rational rational2; // second rational number
14        Rational result; // result of performing an operation
15
16        // read first fraction
17        Console.Write( "Enter numerator 1: " );
18        numerator = Convert.ToInt32( Console.ReadLine() );
19        Console.Write( "Enter denominator 1: " );
20        denominator = Convert.ToInt32( Console.ReadLine() );
21        rational1 = new Rational( numerator, denominator );
22
23        // read second fraction
24        Console.Write( "Enter numerator 2: " );
25        numerator = Convert.ToInt32( Console.ReadLine() );
26        Console.Write( "Enter denominator 2: " );
27        denominator = Convert.ToInt32( Console.ReadLine() );
28        rational2 = new Rational( numerator, denominator );
29
30        Console.Write( "Enter precision: " );
31        digits = Convert.ToInt32( Console.ReadLine() );
32
33        int choice = GetMenuChoice(); // user's choice in the menu
34
35        while ( choice != 5 )
36        {
37            switch ( choice )
38            {
39                case 1:
40                    result = rational1.Sum( rational2 );

```

```

41         Console.WriteLine( "a + b = {0} = {1}",
42             result.ToString(),
43             result.ToFloatString( digits ) );
44         break;
45     case 2:
46         result = rational1.Subtract( rational2 );
47         Console.WriteLine( "a - b = {0} = {1}",
48             result.ToString(),
49             result.ToFloatString( digits ) );
50         break;
51     case 3:
52         result = rational1.Multiply( rational2 );
53         Console.WriteLine( "a * b = {0} = {1}",
54             result.ToString(),
55             result.ToFloatString( digits ) );
56         break;
57     case 4:
58         result = rational1.Divide( rational2 );
59         Console.WriteLine( "a / b = {0} = {1}",
60             result.ToString(),
61             result.ToFloatString( digits ) );
62         break;
63     } // end switch
64
65     choice = GetMenuChoice();
66 } // end while
67 } // end Main
68
69 // displays a menu and returns the chosen value
70 private static int GetMenuChoice()
71 {
72     Console.WriteLine( "1. Add" );
73     Console.WriteLine( "2. Subtract" );
74     Console.WriteLine( "3. Multiply" );
75     Console.WriteLine( "4. Divide" );
76     Console.WriteLine( "5. Exit\n" );
77     Console.Write( "Choice: " );
78
79     return Convert.ToInt32( Console.ReadLine() );
80 } // end method GetMenuChoice
81 } // end class RationalTest

```

```

Enter numerator 1: 5
Enter denominator 1: 10
Enter numerator 2: 2
Enter denominator 2: 13
Enter precision: 4
1. Add
2. Subtract
3. Multiply
4. Divide
5. Exit

```


Choice: 1
 $a + b = 17/26 = 0.6538$

1. Add
2. Subtract
3. Multiply
4. Divide
5. Exit

Choice: 2
 $a - b = 9/26 = 0.3462$

1. Add
2. Subtract
3. Multiply
4. Divide
5. Exit

Choice: 3
 $a * b = 1/13 = 0.0769$

1. Add
2. Subtract
3. Multiply
4. Divide
5. Exit

Choice: 4
 $a / b = 13/4 = 3.2500$

1. Add
2. Subtract
3. Multiply
4. Divide
5. Exit

Choice: 5
 Press any key to continue . . .

10.10 (*HugeInteger* Class) Create a class *HugeInteger* which uses a 40-element array of digits to store integers as large as 40 digits each. Provide methods *Input*, *ToString*, *Add* and *Subtract*. For comparing *HugeInteger* objects, provide the following methods: *IsEqualTo*, *IsNotEqualTo*, *IsGreaterThan*, *IsLessThan*, *IsGreaterThanOrEqualTo* and *IsLessThanOrEqualTo*. Each of these is a method that returns true if the relationship holds between the two *HugeInteger* objects and returns false if the relationship does not hold. Provide method *IsZero*. If you feel ambitious, also provide methods *Multiply*, *Divide* and *Remainder*. In the *Input* method, use the string method *ToCharArray* to convert the input string into an array of characters, then iterate through these characters to create your *HugeInteger*. [Note: The .NET Framework Class Library now includes class *BigInteger* for arbitrary sized integer values.]

ANS:

```

1 // Exercise 10.10 Solution: HugeInteger.cs
2 // HugeInteger class definition
3 public class HugeInteger
4 {
5     private const int DIGITS = 40;
6     private int[] integer; // array containing the integer

```

```

7   private bool positive; // whether the integer is positive
8
9   // parameterless constructor
10  public HugeInteger()
11  {
12      integer = new int[ DIGITS ];
13      positive = true;
14  } // end HugeInteger parameterless constructor
15
16  // convert a string to HugeInteger
17  public void Input( string inputstring )
18  {
19      char[] integerChar = inputstring.ToCharArray();
20
21      // check if input is a negative number
22      if ( integerChar[ 0 ] == '-' )
23          positive = false;
24
25      if ( positive )
26          integer[ DIGITS - integerChar.Length ] =
27              integerChar[ 0 ] - '0';
28
29      // convert string to integer array
30      for ( int i = 1; i < integerChar.Length; i++ )
31          integer[ DIGITS - integerChar.Length + i ] =
32              integerChar[ i ] - '0';
33  } // end method Input
34
35  // add two HugeIntegers
36  public HugeInteger Add( HugeInteger addValue )
37  {
38      HugeInteger temp = new HugeInteger(); // temporary result
39
40      // both HugeInteger are positive or negative
41      if ( positive == addValue.positive )
42          temp = AddPositives( addValue );
43      // addValue is negative
44      else if ( positive && ( !addValue.positive ) )
45      {
46          addValue.positive = true;
47
48          if ( IsGreaterThan( addValue ) )
49              temp = SubtractPositives( addValue );
50          else
51          {
52              temp = addValue.SubtractPositives( this );
53              temp.positive = false;
54          } // end else
55
56          addValue.positive = false; // reset sign for addValue
57      } // end else if
58      // this is negative
59      else if ( !positive && addValue.positive )
60      {

```

```

61         addValue.positive = false;
62
63         if ( IsGreaterThan( addValue ) )
64             temp = addValue.SubtractPositives( this );
65         else
66         {
67             temp = SubtractPositives( addValue );
68             temp.positive = false;
69         } // end else
70
71         addValue.positive = true; // reset sign for addValue
72     } // end else if
73
74     return temp; // return the sum
75 } // end method Add
76
77 // add two positive HugeIntegers
78 public HugeInteger AddPositives( HugeInteger addValue )
79 {
80     HugeInteger temp = new HugeInteger();
81     int carry = 0;
82
83     // iterate through HugeInteger
84     for ( int i = DIGITS - 1; i >= 0; i-- )
85     {
86         temp.integer[ i ] =
87             integer[ i ] + addValue.integer[ i ] + carry;
88
89         // determine whether to carry a 1
90         if ( temp.integer[ i ] > 9 )
91         {
92             temp.integer[ i ] %= 10; // reduce to 0-9
93             carry = 1;
94         } // end if
95         else
96             carry = 0;
97     } // end for
98
99     // if both are negative, set the result to negative
100    if ( !positive )
101        temp.positive = false;
102
103    return temp;
104 } // end method AddPositives
105
106 // subtract two HugeIntegers
107 public HugeInteger Subtract( HugeInteger subtractValue )
108 {
109     HugeInteger temp = new HugeInteger(); // temporary result
110
111     // subtractValue is negative
112     if ( positive && ( !subtractValue.positive ) )
113         temp = AddPositives( subtractValue );

```

```

114 // this HugeInteger is negative
115 else if ( !positive && subtractValue.positive )
116     temp = AddPositives( subtractValue );
117 // at this point, both HugeIntegers have the same sign
118 else
119 {
120     bool isPositive = positive; // original sign
121     bool resultPositive = positive; // sign of the result
122
123     // set both to positive so we can compare absolute values
124     positive = true;
125     subtractValue.positive = true;
126
127     if ( this.IsGreaterThan( subtractValue ) )
128         temp = this.SubtractPositives( subtractValue );
129     else
130     {
131         temp = subtractValue.SubtractPositives( this );
132         resultPositive = !isPositive; // flip the sign
133     } // end else
134
135     positive = isPositive;
136     subtractValue.positive = isPositive;
137     temp.positive = resultPositive;
138 } // end else
139
140 return temp;
141 } // end method Subtract
142
143 // subtract two positive HugeIntegers
144 public HugeInteger SubtractPositives( HugeInteger subtractValue )
145 {
146     HugeInteger temp = new HugeInteger();
147
148     // iterate through HugeInteger
149     for ( int i = DIGITS - 1; i >= 0; i-- )
150     {
151         // borrow if needed
152         if ( integer[ i ] < subtractValue.integer[ i ] )
153         {
154             integer[ i ] += 10;
155             subtractValue.integer[ i - 1 ]--;
156         } // end if
157
158         temp.integer[ i ] = integer[ i ] - subtractValue.integer[ i ];
159     } // end for
160
161     return temp; // return difference of two HugeIntegers
162 } // end method SubtractPositives
163
164 // find first non-zero position of HugeInteger
165 public int FindFirstNonZeroPosition()
166 {

```

```

167 // find first non-zero position for HugeInteger
168 for ( int i = 0; i < DIGITS; i++ )
169 {
170     if ( integer[ i ] > 0 )
171         return i;
172 } // end for
173
174 return DIGITS - 1;
175 } // end method FindFirstNonZeroPosition
176
177 // get string representation of HugeInteger
178 public override string ToString()
179 {
180     string output = string.Empty;
181
182     if ( !positive )
183         output = "-";
184
185     // get HugeInteger values without leading zeros
186     for ( int i = FindFirstNonZeroPosition(); i < DIGITS; i++ )
187         output += integer[ i ];
188
189     return output;
190 } // end method ToString
191
192 // test if two HugeIntegers are equal
193 public bool IsEqualTo( HugeInteger compareValue )
194 {
195     // compare the sign
196     if ( positive != compareValue.positive )
197         return false;
198
199     // compare each digit
200     for ( int i = 0; i < DIGITS; i++ )
201     {
202         if ( integer[ i ] != compareValue.integer[ i ] )
203             return false;
204     } // end for
205
206     return true;
207 } // end method IsEqualTo
208
209 // test if two HugeIntegers are not equal
210 public bool IsNotEqualTo( HugeInteger compareValue )
211 {
212     return !IsEqualTo( compareValue );
213 } // end method IsNotEqualTo
214
215 // test if one HugeInteger is greater than another
216 public bool IsGreaterThan( HugeInteger compareValue )
217 {
218     // different signs
219     if ( positive && ( !compareValue.positive ) )
220         return true;

```

```

221     else if ( !positive && compareValue.positive )
222         return false;
223
224     // same sign
225     else
226     {
227         // first number's Length is less than second number's Length
228         if ( FindFirstNonZeroPosition() >
229             compareValue.FindFirstNonZeroPosition() )
230         {
231             return !positive;
232         } // end if
233
234         // first number's Length is larger than that of second number
235         else if ( FindFirstNonZeroPosition() <
236                 compareValue.FindFirstNonZeroPosition() )
237         {
238             return positive;
239         } // end else if
240
241         // two numbers have same Length
242         else
243         {
244             for ( int i = 0; i < DIGITS; i++ )
245             {
246                 if ( integer[ i ] > compareValue.integer[ i ] )
247                     return positive;
248                 else if ( integer[ i ] < compareValue.integer[ i ] )
249                     return !positive;
250             } // end for
251         } // end else
252     } // end outer if-elseif-else
253
254     return false;
255 } // end method IsGreaterThan
256
257 // test if one HugeInteger is less than another
258 public bool IsLessThan( HugeInteger compareValue )
259 {
260     return !( IsGreaterThan( compareValue ) ||
261              IsEqualTo( compareValue ) );
262 } // end method IsLessThan
263
264 // test if one HugeInteger is great than or equal to another
265 public bool IsGreaterThanOrEqualTo( HugeInteger compareValue )
266 {
267     return !IsLessThan( compareValue );
268 } // end method IsGreaterThanOrEqualTo
269
270 // test if one HugeInteger is less than or equal to another
271 public bool IsLessThanOrEqualTo( HugeInteger compareValue )
272 {
273     return !IsGreaterThan( compareValue );
274 } // end method IsLessThanOrEqualTo

```

```

275
276 // test if one HugeInteger is zero
277 public bool IsZero()
278 {
279     // compare each digit
280     for ( int i = 0; i < DIGITS; i++ )
281     {
282         if ( integer[ i ] != 0 )
283             return false;
284     } // end for
285
286     return true;
287 } // end method IsZero
288 } // end class HugeInteger

```

```

1 // Exercise 10.10 Solution: HugeIntegerTest.cs
2 // Test class HugeInteger
3 using System;
4
5 public class HugeIntegerTest
6 {
7     public static void Main( string[] args )
8     {
9         HugeInteger integer1 = new HugeInteger();
10        HugeInteger integer2 = new HugeInteger();
11
12        Console.Write( "Enter first HugeInteger: " );
13        integer1.Input( Console.ReadLine() );
14
15        Console.Write( "Enter second HugeInteger: " );
16        integer2.Input( Console.ReadLine() );
17
18        Console.WriteLine( "HugeInteger 1: {0}", integer1.ToString() );
19        Console.WriteLine( "HugeInteger 2: {0}", integer2.ToString() );
20
21        HugeInteger result;
22
23        // add two HugeIntegers
24        result = integer1.Add( integer2 );
25        Console.WriteLine( "Add result: {0}", result.ToString() );
26
27        // subtract two HugeIntegers
28        result = integer1.Subtract( integer2 );
29        Console.WriteLine( "Subtract result: {0}", result.ToString() );
30
31        // compare two HugeIntegers
32        Console.WriteLine(
33            "HugeInteger 1 is zero: {0}", integer1.IsZero() );
34        Console.WriteLine(
35            "HugeInteger 2 is zero: {0}", integer2.IsZero() );
36        Console.WriteLine(
37            "HugeInteger 1 is equal to HugeInteger 2: {0}",
38            integer1.IsEqualTo( integer2 ) );

```

```

39     Console.WriteLine(
40         "HugeInteger 1 is not equal to HugeInteger 2: {0}",
41         integer1.IsNotEqualTo( integer2 ) );
42     Console.WriteLine(
43         "HugeInteger 1 is greater than HugeInteger 2: {0}",
44         integer1.IsGreaterThan( integer2 ) );
45     Console.WriteLine(
46         "HugeInteger 1 is less than HugeInteger 2: {0}",
47         integer1.IsLessThan( integer2 ) );
48     Console.WriteLine(
49         "HugeInteger 1 is greater than or equal to HugeInteger 2: {0}",
50         integer1.IsGreaterThanOrEqualTo( integer2 ) );
51     Console.WriteLine(
52         "HugeInteger 1 is less than or equal to HugeInteger 2: {0}",
53         integer1.IsLessThanOrEqualTo( integer2 ) );
54 } // end Main
55 } // end class HugeIntegerTest

```

```

Enter first HugeInteger: -1234567890987654321
Enter second HugeInteger: 1111111111111111111
HugeInteger 1: -1234567890987654321
HugeInteger 2: 1111111111111111111
Add result: -123456799876543210
Subtract result: -2345679002098765432
HugeInteger 1 is zero: False
HugeInteger 2 is zero: False
HugeInteger 1 is equal to HugeInteger 2: False
HugeInteger 1 is not equal to HugeInteger 2: True
HugeInteger 1 is greater than HugeInteger 2: False
HugeInteger 1 is less than HugeInteger 2: True
HugeInteger 1 is greater than or equal to HugeInteger 2: False
HugeInteger 1 is less than or equal to HugeInteger 2: True

```

10.11 (*Tic-Tac-Toe*) Create class `TicTacToe` that will enable you to write a complete application to play the game of Tic-Tac-Toe. The class contains a private 3-by-3 rectangular array of integers. The constructor should initialize the empty board to all 0s. Allow two human players. Wherever the first player moves, place a 1 in the specified square, and place a 2 wherever the second player moves. Each move must be to an empty square. After each move determine whether the game has been won and whether it is a draw. If you feel ambitious, modify your application so that the computer makes the moves for one of the players. Also, allow the player to specify whether he or she wants to go first or second. If you feel exceptionally ambitious, develop an application that will play three-dimensional Tic-Tac-Toe on a 4-by-4-by-4 board.

ANS:

```

1 // Exercise 10.11 Solution: TicTacToe.cs
2 // Tic-tac-toe.
3 using System;
4
5 public class TicTacToe
6 {
7     public enum Status { WIN, DRAW, CONTINUE }; // game states
8

```

```

 9  private const int BOARDSIZE = 3; // size of the board
10  private int[ , ] board; // board representation
11  private bool firstPlayer; // whether it's player 1's move
12  private bool gameOver; // whether the game is over
13
14  // Constructor
15  public TicTacToe()
16  {
17      board = new int[ BOARDSIZE, BOARDSIZE ];
18      firstPlayer = true;
19      gameOver = false;
20  } // end Constructor
21
22  // start game
23  public void Play()
24  {
25      int row; // row for next move
26      int column; // column for next move
27
28      Console.WriteLine( "\nPlayer 1's turn." );
29
30      while ( !gameOver )
31      {
32          int player = ( firstPlayer ? 1 : 2 );
33          // first player's turn
34
35          do
36          {
37              Console.Write(
38                  "Player {0}: Enter row ( 0 <= row < 3 ): ", player );
39              row = Convert.ToInt32( Console.ReadLine() );
40              Console.Write(
41                  "Player {0}: Enter column ( 0 <= row < 3 ): ", player );
42              column = Convert.ToInt32( Console.ReadLine() );
43          } while ( !ValidMove( row, column ) );
44
45          board[ row, column ] = player;
46
47          firstPlayer = !firstPlayer;
48
49          PrintBoard();
50          Console.WriteLine();
51          PrintStatus( player );
52      } // end while
53  } // end method Play
54
55  // show game status in status bar
56  public void PrintStatus( int player )
57  {
58      Status status = GameStatus();
59
60      // check game status
61      switch ( status )
62      {

```

```

63         case Status.WIN:
64             Console.WriteLine( "Player " + player + " wins." );
65             gameOver = true;
66             break;
67         case Status.DRAW:
68             Console.WriteLine( "Game is a draw." );
69             gameOver = true;
70             break;
71         case Status.CONTINUE:
72             if ( player == 1 )
73                 Console.WriteLine( "Player 2's turn." );
74             else
75                 Console.WriteLine( "Player 1's turn." );
76             break;
77     } // end switch
78 } // end method PrintStatus
79
80 // get game status
81 public Status GameStatus()
82 {
83     int a;
84
85     // check for a win on diagonals
86     if ( board[ 0, 0 ] != 0 && board[ 0, 0 ] == board[ 1, 1 ] &&
87         board[ 0, 0 ] == board[ 2, 2 ] )
88         return Status.WIN;
89     else if ( board[ 2, 0 ] != 0 && board[ 2, 0 ] ==
90         board[ 1, 1 ] && board[ 2, 0 ] == board[ 0, 2 ] )
91         return Status.WIN;
92
93     // check for win in rows
94     for ( a = 0; a < 3; a++ )
95         if ( board[ a, 0 ] != 0 && board[ a, 0 ] ==
96             board[ a, 1 ] && board[ a, 0 ] == board[ a, 2 ] )
97             return Status.WIN;
98
99     // check for win in columns
100    for ( a = 0; a < 3; a++ )
101        if ( board[ 0, a ] != 0 && board[ 0, a ] ==
102            board[ 1, a ] && board[ 0, a ] == board[ 2, a ] )
103            return Status.WIN;
104
105    // check for a completed game
106    for ( int r = 0; r < 3; r++ )
107        for ( int c = 0; c < 3; c++ )
108            if ( board[ r, c ] == 0 )
109                return Status.CONTINUE; // game is not finished
110
111    return Status.DRAW; // game is a draw
112 } // end method GameStatus
113
114 // display board
115 public void PrintBoard()
116 {

```

```

117 Console.WriteLine( " _____ " );
118
119 for ( int row = 0; row < BOARDSIZE; row++ )
120 {
121     Console.WriteLine( "|      |      |      |" );
122
123     for ( int column = 0; column < BOARDSIZE; column++ )
124         PrintSymbol( column, board[ row, column ] );
125
126     Console.WriteLine( "|_____|_____|_____|" );
127 } // end for
128 } // end method PrintBoard
129
130 // display moves
131 public void PrintSymbol( int column, int player )
132 {
133     switch ( player )
134     {
135         case 0:
136             Console.Write( "|      " );
137             break;
138         case 1:
139             Console.Write( "|  1  " );
140             break;
141         case 2:
142             Console.Write( "|  2  " );
143             break;
144     } // end switch
145
146     if ( column == 2 )
147         Console.WriteLine( "|" );
148 } // end method PrintSymbol
149
150 // validate move
151 public bool ValidMove( int row, int column )
152 {
153     return row >= 0 && row < BOARDSIZE && column >= 0 &&
154         column < BOARDSIZE && board[ row, column ] == 0;
155 } // end method ValidMove
156 } // end class TicTacToe

```

```

1 // Exercise 10.11 Solution: TicTacToeTest.cs
2 // Play a game of Tic Tac Toe
3 public class TicTacToeTest
4 {
5     public static void Main( string[] args )
6     {
7         TicTacToe game = new TicTacToe();
8         game.PrintBoard();
9         game.Play();
10    } // end Main
11 } // end class TicTacToeTest

```

Player 1's turn.

Player 1: Enter row ($0 \leq \text{row} < 3$): 1

Player 1: Enter column ($0 \leq \text{row} < 3$): 1

	1	

Player 2's turn.

Player 2: Enter row ($0 \leq \text{row} < 3$): 1

Player 2: Enter column ($0 \leq \text{row} < 3$): 2

	1	2

Player 1's turn.

Player 1: Enter row ($0 \leq \text{row} < 3$): 2

Player 1: Enter column ($0 \leq \text{row} < 3$): 1

	1	2
	1	

Player 2's turn.

Player 2: Enter row ($0 \leq \text{row} < 3$): 0

Player 2: Enter column ($0 \leq \text{row} < 3$): 1

	2	
	1	2
	1	

Player 1's turn.

Player 1: Enter row ($0 \leq \text{row} < 3$): 2

Player 1: Enter column ($0 \leq \text{row} < 3$): 2

	2	
	1	2
	1	1

Player 2's turn.

Player 2: Enter row ($0 \leq \text{row} < 3$): 0

Player 2: Enter column ($0 \leq \text{row} < 3$): 0

2	2	
	1	2
	1	1

Player 1's turn.

Player 1: Enter row ($0 \leq \text{row} < 3$): 2

Player 1: Enter column ($0 \leq \text{row} < 3$): 0

2	2	
	1	2
1	1	1

Player 1 wins.

10.12 What happens when a return type, even `void`, is specified for a constructor?

ANS: It is a compilation error. *C#* will assume that this is a method declaration instead of a constructor declaration, and methods cannot have the same name as their class.

11

Object-Oriented Programming: Inheritance

Say not you know another entirely, till you have divided an inheritance with him.

—Johann Kasper Lavater

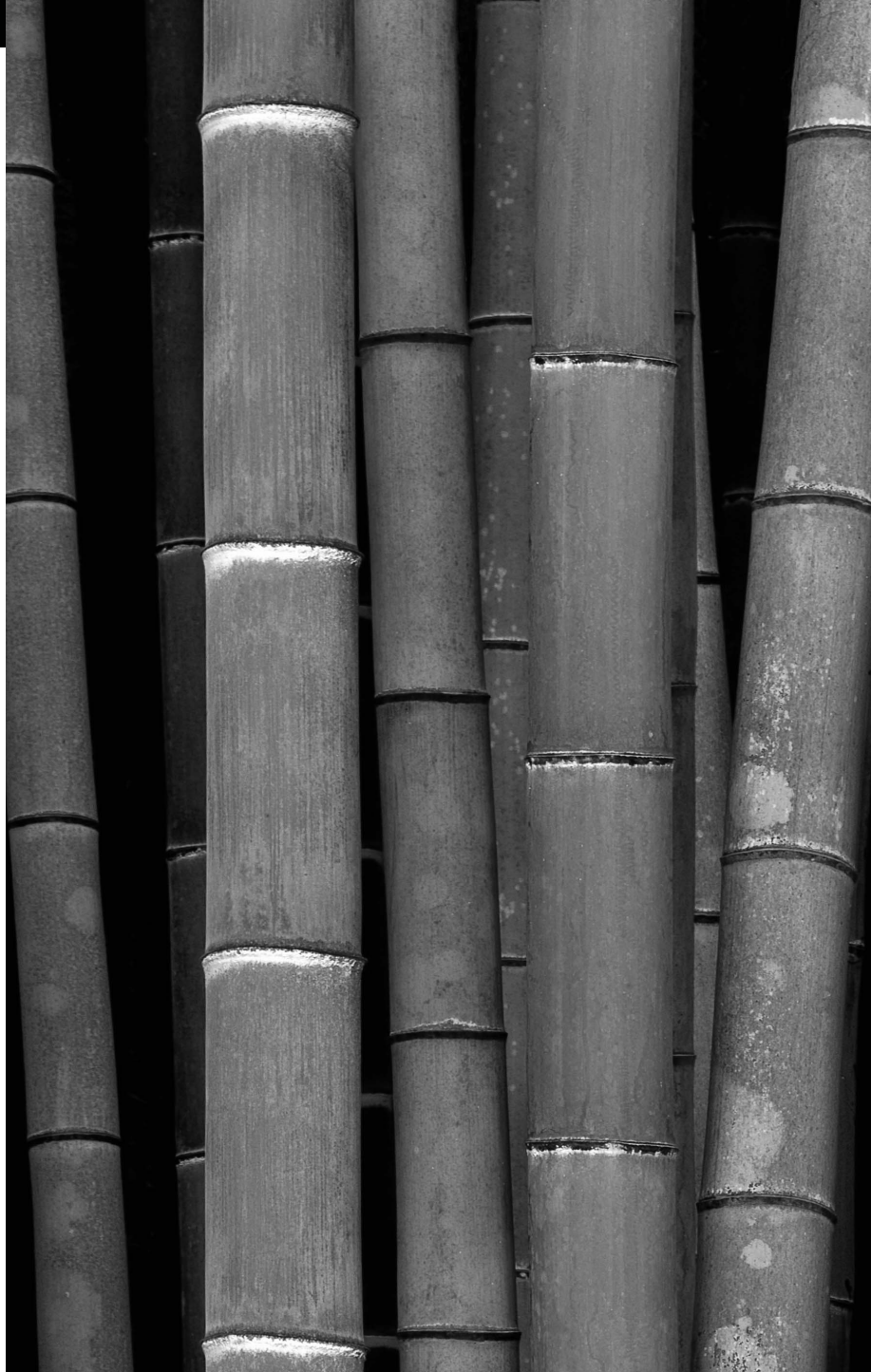
This method is to define as the number of a class the class of all classes similar to the given class.

—Bertrand Russell

Objectives

In this chapter you'll learn:

- How inheritance promotes software reusability.
- To create a derived class that inherits attributes and behaviors from a base class.
- To use access modifier **protected** to give derived-class methods access to base-class members.
- To access base-class members with **base**.
- How constructors are used in inheritance hierarchies.
- The methods of class **object**, the direct or indirect base class of all classes.



Self-Review Exercises

11.1 Fill in the blanks in each of the following statements:

- a) _____ is a form of software reusability in which new classes acquire the members of existing classes and enhance those classes with new capabilities.

ANS: Inheritance

- b) A base class's _____ members can be accessed only in the base class declaration and in derived class declarations.

ANS: protected

- c) In a(n) _____ relationship, an object of a derived class can also be treated as an object of its base class.

ANS: *is-a* or inheritance

- d) In a(n) _____ relationship, a class object has references to objects of other classes as members.

ANS: "has-a" or composition

- e) In single inheritance, a base class exists in a(n) _____ relationship with its derived classes.

ANS: hierarchical

- f) A base class's _____ members are accessible anywhere that the application has a reference to an object of that base class or to an object of any of its derived classes.

ANS: public

- g) When an object of a derived class is instantiated, a base class _____ is called implicitly or explicitly.

ANS: constructor

- h) Derived class constructors can call base class constructors via the _____ keyword.

ANS: base

11.2 State whether each of the following is *true* or *false*. If a statement is *false*, explain why.

- a) Base class constructors are not inherited by derived classes.

ANS: True.

- b) A *has-a* relationship is implemented via inheritance.

ANS: False. A *has-a* relationship is implemented via composition. An *is-a* relationship is implemented via inheritance.

- c) A Car class has *is-a* relationships with the SteeringWheel and Brakes classes.

ANS: False. These are examples of *has-a* relationships. Class Car has an *is-a* relationship with class Vehicle.

- d) Inheritance encourages the reuse of proven high-quality software.

ANS: True.

- e) When a derived class redefines a base class method by using the same signature and return type, the derived class is said to overload that base class method.

ANS: False. This is known as overriding, not overloading.

Exercises

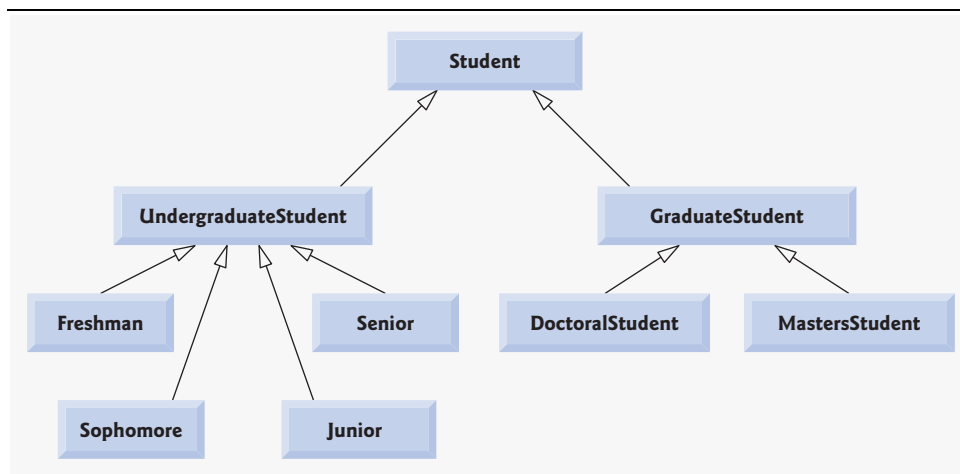
NOTE: Solutions to the programming exercises are located in the `sol_ch11` folder. Each exercise has its own folder named `ex11_##` where `##` is a two-digit number representing the exercise number. For example, Exercise 11.3's solution is located in the folder `ex11_03`.

11.4 (Inheritance and Software Reuse) Discuss the ways in which inheritance promotes software reuse, saves time during application development and helps prevent errors.

ANS: Inheritance allows you to create derived classes that reuse code declared already in a base class. Avoiding the duplication of common functionality between several classes by building an inheritance hierarchy to contain the classes can save you a considerable amount of time. Similarly, placing common functionality in a single base class, rather than duplicating the code in multiple unrelated classes, helps prevent the same errors from appearing in multiple source code files. If several classes each contain duplicate code containing an error, you have to spend time correcting each source code file with the error. However, if these classes take advantage of inheritance, and the error occurs in the common functionality of the base class, you need to modify only the base class's source code file.

11.5 (Student Inheritance Hierarchy) Draw a UML class diagram for an inheritance hierarchy for students at a university similar to the hierarchy shown in Fig. 11.2. Use `Student` as the base class of the hierarchy, then extend `Student` with classes `UndergraduateStudent` and `GraduateStudent`. Continue to extend the hierarchy as deeply (i.e., as many levels) as possible. For example, `Freshman`, `Sophomore`, `Junior` and `Senior` might extend `UndergraduateStudent`, and `DoctoralStudent` and `MastersStudent` might be derived classes of `GraduateStudent`. After drawing the hierarchy, discuss the relationships that exist between the classes. [Note: You do not need to write any code for this exercise.]

ANS:

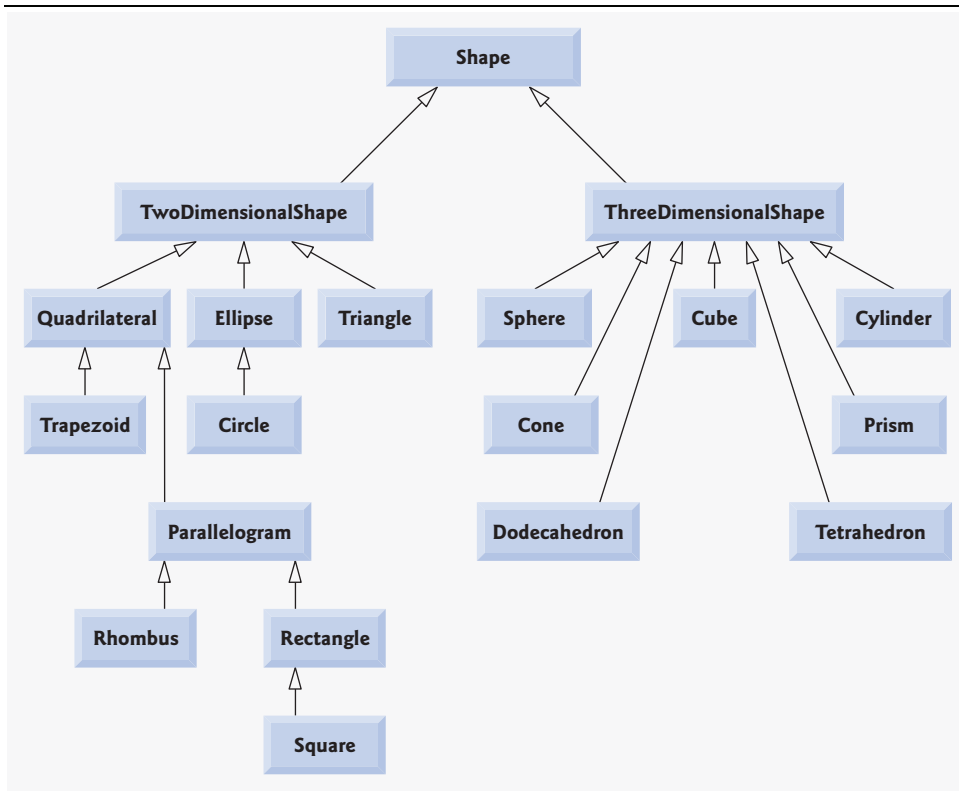


This hierarchy contains many “is-a” (inheritance) relationships. An `UndergraduateStudent` *is* a `Student`. A `GraduateStudent` *is* a `Student` too. Each of the classes `Freshman`, `Sophomore`, `Junior` and `Senior` *is* an `UndergraduateStudent` and *is* a `Student`. Each of the classes `DoctoralStudent` and `MastersStudent` *is* a `GraduateStudent` and *is* a `Student`.

11.6 (Shape Inheritance Hierarchy) The world of shapes is much richer than the shapes included in the inheritance hierarchy of Fig. 11.3. Write down all the shapes you can think of—both two-dimensional and three-dimensional—and form them into a more complete Shape hierarchy with as many levels as possible. Your hierarchy should have class `Shape` at the top. Class `TwoDimension-`

alShape and class ThreeDimensionalShape should extend Shape. Add additional derived classes, such as Quadrilateral and Sphere, at their correct locations in the hierarchy as necessary.

ANS: [Note: Solutions may vary.]



11.7 (Protected vs. Private Access) Some programmers prefer not to use protected access, because they believe it breaks the encapsulation of the base class. Discuss the relative merits of using protected access vs. using private access in base classes.

ANS: private instance variables are hidden in the derived class and are accessible only through the public or protected properties and methods of the base class. Using protected access enables the derived class to manipulate the protected members without using the properties and methods of the base class. If the base class instance variables are private, the properties of the base class must be used to access the data. This may result in a decrease in performance due to the extra accessor and method calls. Declaring private instance variables in a base class helps you test, debug and correctly modify systems. If a derived class could access its base class's private instance variables, classes that inherit from that base class could access the instance variables as well. This would propagate access to what should be private instance variables, and the benefits of information hiding would be lost.

12

Polymorphism, Interfaces and Operator Overloading: Solutions

*One Ring to rule them all,
One Ring to find them,
One Ring to bring them all and
in the darkness bind them.*

—John Ronald Reuel Tolkien

*General propositions do not
decide concrete cases.*

—Oliver Wendell Holmes

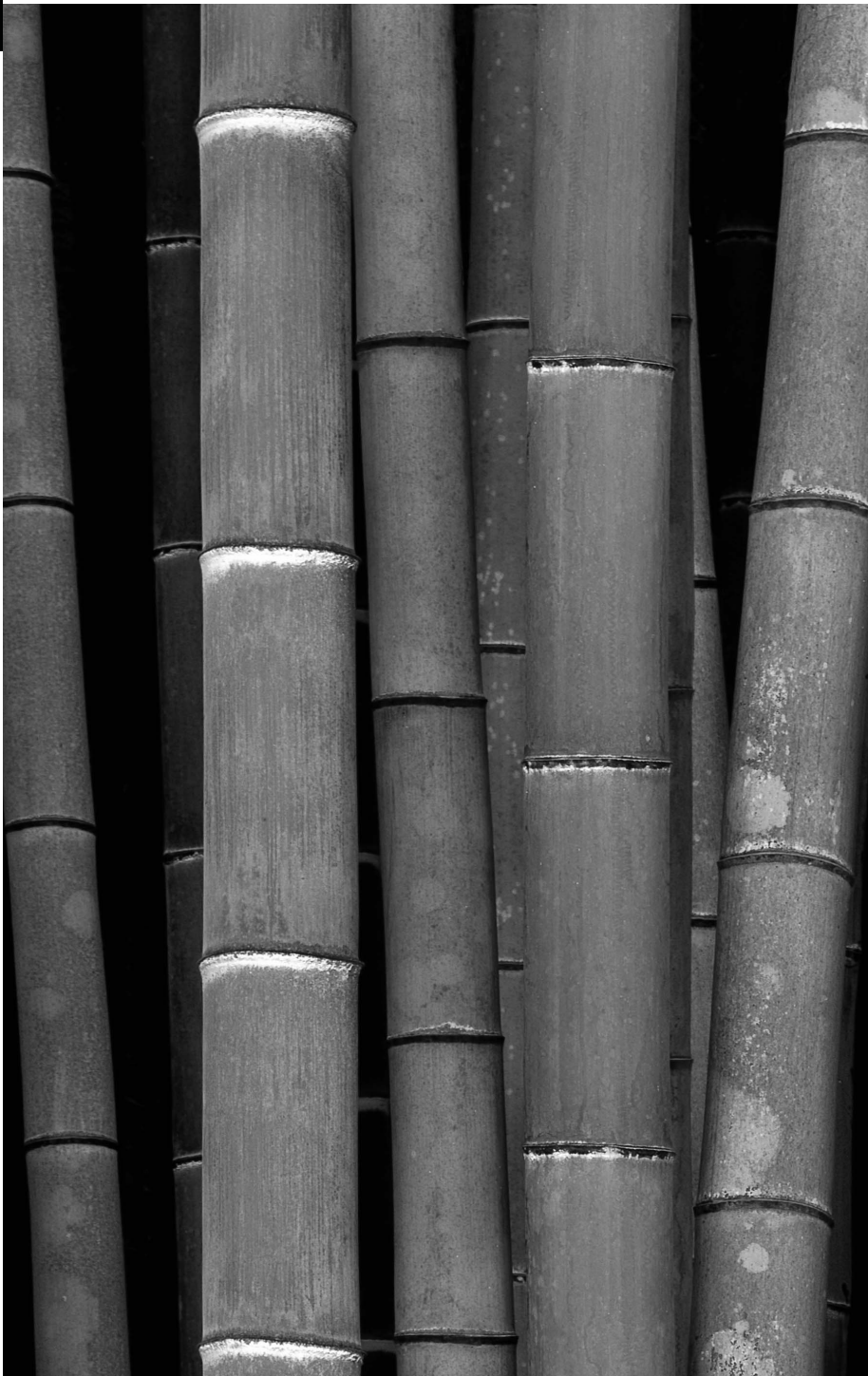
*A philosopher of imposing
stature doesn't think in a
vacuum. Even his most abstract
ideas are, to some extent,
conditioned by
what is or is not known
in the time when he lives.*

—Alfred North Whitehead

Objectives

In this chapter you'll learn:

- How polymorphism enables you to “program in the general” and make systems extensible.
- To use overridden methods to effect polymorphism.
- To create abstract classes and methods.
- To determine an object's type at execution time.
- To create `sealed` methods and classes.
- To declare and implement interfaces.
- To overload operators to enable them to manipulate objects.



Self-Review Exercises

12.1 Fill in the blanks in each of the following statements:

- a) If a class contains at least one abstract method, it must be declared as a(n) _____ class.

ANS: abstract

- b) Classes from which objects can be instantiated are called _____ classes.

ANS: concrete

- c) _____ involves using a base class variable to invoke methods on base class and derived class objects, enabling you to “program in the general.”

ANS: Polymorphism

- d) Methods in a class that do not provide implementations must be declared using keyword _____.

ANS: abstract

- e) Casting a reference stored in a base class variable to a derived class type is called _____.

ANS: downcasting

12.2 State whether each of the following statements is *true* or *false*. If *false*, explain why.

- a) It is possible to treat base class objects and derived class objects similarly.

ANS: True.

- b) All methods in an abstract class must be declared as abstract methods.

ANS: False. An abstract class can include methods with implementations and abstract methods.

- c) Attempting to invoke a derived-class-only method through a base-class variable is an error.

ANS: True.

- d) If a base class declares an abstract method, a derived class must implement that method.

ANS: False. Only a concrete derived class must implement the method.

- e) An object of a class that implements an interface may be thought of as an object of that interface type.

ANS: True.

Exercises

12.3 (*Programming in the General*) How does polymorphism enable you to program “in the general” rather than “in the specific”? Discuss the key advantages of programming “in the general.”

ANS: Polymorphism enables the programmer to concentrate on the common operations that are applied to objects of all the classes below a particular base class in a hierarchy. The general processing capabilities can be separated from any code that is specific to each class. Those general portions of the code can accommodate new classes without modification. In some polymorphic applications, only the code that creates the objects needs to be modified to extend the system with new classes.

12.4 (*Inheriting Interface vs. Inheriting Implementation*) A derived class can inherit “interface” or “implementation” from a base class. How do inheritance hierarchies designed for inheriting interface differ from those designed for inheriting implementation?

ANS: Hierarchies designed for implementation inheritance tend to define their functionality high in the hierarchy—each new derived class inherits one or more methods that were declared in a base class, and the derived class uses the base class implementations (sometimes overriding the base class methods and calling them as part of the derived class implementations). Hierarchies designed for interface inheritance tend to have

their functionality defined lower in the hierarchy—a base class specifies one or more abstract methods that must be declared for each class in the hierarchy, and the individual derived classes override these methods to provide derived-class-specific implementations.

12.5 (*Abstract Methods*) What are abstract methods? Describe the circumstances in which an abstract method would be appropriate.

ANS: An abstract method is one with keyword `abstract` in its declaration. Abstract methods do not provide implementations. Each concrete derived class of an abstract base class must provide concrete implementations of the base class's abstract methods. An abstract method is appropriate when it does not make sense to provide an implementation for a method in a base class (i.e., some additional derived-class-specific knowledge is required to implement the method in a meaningful manner).

12.6 (*Polymorphism and Extensibility*) How does polymorphism promote extensibility?

ANS: Software that invokes polymorphic behavior is independent of the object types to which messages are sent. New object types that can respond to existing method calls can be incorporated into a system without requiring modification of the base system. Only client code that instantiates new objects must be modified to accommodate new types.

12.7 (*Assigning Base Class and Derived Class References*) Discuss four ways in which you can assign base-class and derived-class references to variables of base-class and derived-class types.

ANS: 1) Assigning a base class reference to a base class variable. 2) Assigning a derived class reference to a derived class variable. 3) Assigning a derived class object's reference to a base class variable is safe, because the derived class object *is an* object of its base class. However, this reference can be used to refer only to base class members. If this code refers to derived-class-only members through the base class variable, the compiler reports errors. 4) Attempting to assign a base class object's reference to a derived class variable is a compilation error. To avoid this error, the base class reference must be downcast to a derived class type explicitly. At execution time, if the object to which the reference refers is not a derived class object, an exception will occur. The `is` operator can be used to ensure that such a cast is performed only if the object is a derived class object. You can also use the `as` operator, which will return null if the object is not the correct type.

12.8 (*Abstract Classes vs. Interfaces*) Compare and contrast abstract classes and interfaces. Why would you use an abstract class? Why would you use an interface?

ANS: An abstract class describes the general notion of what it means to be an object of that class. Abstract classes are incomplete—they normally contain data and one or more methods and properties that are declared abstract because they cannot be implemented in a general sense. Objects of an abstract class cannot be instantiated. Concrete derived classes must declare the “missing pieces”—the implementations of the abstract methods and properties that are appropriate for each specific derived class. Abstract class references can refer to objects of any of the classes below the abstract class in an inheritance hierarchy and therefore can be used to process any such objects polymorphically. An interface also describes abstract functionality that can be implemented by objects of any number of classes. Classes that implement an interface can be completely unrelated, whereas concrete derived classes of the same abstract base class are all related to one other by way of a shared base class. An interface is often used when disparate (i.e., unrelated) classes need to provide common functionality (i.e., methods and properties). An interface can also be used in place of an abstract class when there are no default implementation details (i.e., method and property im-

plementations and instance variables) to inherit. When a class implements an interface, it establishes an *is-a* relationship with the interface type, just as a derived class participates in an *is-a* relationship with its base class. Therefore, interface references can be used to evoke polymorphic behavior, just as abstract base class references can.

12.9 (Payroll System Modification) Modify the payroll system of Figs. 12.4–12.9 to include private instance variable `birthDate` in class `Employee`. Use class `Date` of Fig. 10.7 to represent an employee's birthday. Assume that payroll is processed once per month. Create an array of `Employee` variables to store references to the various employee objects. In a loop, calculate the payroll for each `Employee` (polymorphically), and add a \$100.00 bonus to the person's payroll amount if the current month is the month in which the `Employee`'s birthday occurs.

ANS:

```

1  // Exercise 12.9 Solution: Employee.cs
2  // Employee abstract base class.
3  public abstract class Employee
4  {
5      // auto-implemented read-only property FirstName
6      public string FirstName { get; private set; }
7
8      // auto-implemented read-only property LastName
9      public string LastName { get; private set; }
10
11     // auto-implemented read-only property SocialSecurityNumber
12     public string SocialSecurityNumber { get; private set; }
13
14     // auto-implemented read-only property BirthDate
15     public Date BirthDate { get; private set; }
16
17     // six-parameter constructor
18     public Employee( string first, string last, string ssn,
19                     int month, int day, int year )
20     {
21         FirstName = first;
22         LastName = last;
23         SocialSecurityNumber = ssn;
24         BirthDate = new Date( month, day, year );
25     } // end six-parameter Employee constructor
26
27     // return string representation of Employee object
28     public override string ToString()
29     {
30         return string.Format( "{0} {1}\n{2}: {3}\n{4}: {5}",
31                               FirstName, LastName,
32                               "social security number", SocialSecurityNumber,
33                               "birth date", BirthDate );
34     } // end method ToString
35
36     // abstract method overridden by derived classes
37     public abstract decimal Earnings();
38 } // end abstract class Employee

```

```

1  // Exercise 12.9 Solution: CommissionEmployee.cs
2  // CommissionEmployee class that extends Employee.
3  using System;
4
5  public class CommissionEmployee : Employee
6  {
7      private decimal grossSales; // gross weekly sales
8      private decimal commissionRate; // commission percentage
9
10     // eight-parameter constructor
11     public CommissionEmployee( string first, string last, string ssn,
12         int month, int day, int year, decimal sales, decimal rate )
13         : base( first, last, ssn, month, day, year )
14     {
15         GrossSales = sales; // validate gross sales via property
16         CommissionRate = rate; // validate commission rate via property
17     } // end eight-parameter CommissionEmployee constructor
18
19     // property that gets and sets commission employee's gross sales
20     public decimal GrossSales
21     {
22         get
23         {
24             return grossSales;
25         } // end get
26         set
27         {
28             if ( value >= 0 )
29                 grossSales = value;
30             else
31                 throw new ArgumentOutOfRangeException(
32                     "GrossSales", value, "GrossSales must be >= 0" );
33         } // end set
34     } // end property GrossSales
35
36     // property that gets and sets commission employee's commission rate
37     public decimal CommissionRate
38     {
39         get
40         {
41             return commissionRate;
42         } // end get
43         set
44         {
45             if ( value > 0 && value < 1 )
46                 commissionRate = value;
47             else
48                 throw new ArgumentOutOfRangeException( "CommissionRate",
49                     value, "CommissionRate must be > 0 and < 1" );
50         } // end set
51     } // end property CommissionRate
52
53     // calculate earnings; override abstract method Earnings in Employee
54     public override decimal Earnings()
55     {

```

```

56     return CommissionRate * GrossSales;
57 } // end method Earnings
58
59 // return string representation of CommissionEmployee object
60 public override string ToString()
61 {
62     return string.Format( "{0}: {1}\n{2}: {3:C}; {4}: {5:F}",
63         "commission employee", base.ToString(),
64         "gross sales", GrossSales,
65         "commission rate", CommissionRate );
66 } // end method ToString
67 } // end class CommissionEmployee

```

```

1  // Exercise 12.9 Solution: BasePlusCommissionEmployee.cs
2  // BasePlusCommissionEmployee class that extends CommissionEmployee.
3  using System;
4
5  public class BasePlusCommissionEmployee : CommissionEmployee
6  {
7      private decimal baseSalary; // base salary per week
8
9      // nine-parameter constructor
10     public BasePlusCommissionEmployee( string first, string last,
11         string ssn, int month, int day, int year,
12         decimal sales, decimal rate, decimal salary )
13         : base( first, last, ssn, month, day, year, sales, rate )
14     {
15         BaseSalary = salary; // validate base salary via property
16     } // end nine-parameter BasePlusCommissionEmployee constructor
17
18     // property that gets and sets
19     // base-salaried commission employee's base salary
20     public decimal BaseSalary
21     {
22         get
23         {
24             return baseSalary;
25         } // end get
26         set
27         {
28             if ( value >= 0 )
29                 baseSalary = value;
30             else
31                 throw new ArgumentOutOfRangeException( "BaseSalary",
32                     value, "BaseSalary must be >= 0" );
33         } // end set
34     } // end property BaseSalary
35
36     // calculate earnings; override method Earnings in CommissionEmployee
37     public override decimal Earnings()
38     {
39         return BaseSalary + base.Earnings();
40     } // end method Earnings

```

```

41
42 // return string representation of BasePlusCommissionEmployee object
43 public override string ToString()
44 {
45     return string.Format( "{0} {1}; {2}: {3:C}",
46         "base-salaried", base.ToString(),
47         "base salary", BaseSalary );
48 } // end method ToString
49 } // end class BasePlusCommissionEmployee

```

```

1 // Exercise 12.9 Solution: HourlyEmployee.cs
2 // HourlyEmployee class that extends Employee.
3 using System;
4
5 public class HourlyEmployee : Employee
6 {
7     private decimal wage; // wage per hour
8     private decimal hours; // hours worked for week
9
10    // eight-parameter constructor
11    public HourlyEmployee( string first, string last, string ssn,
12        int month, int day, int year,
13        decimal hourlyWage, decimal hoursWorked )
14        : base( first, last, ssn, month, day, year )
15    {
16        Wage = hourlyWage;
17        Hours = hoursWorked;
18    } // end eight-parameter HourlyEmployee constructor
19
20    // property that gets and sets hourly employee's wage
21    public decimal Wage
22    {
23        get
24        {
25            return wage;
26        } // end get
27        set
28        {
29            if ( value >= 0 ) // validation
30                wage = value;
31            else
32                throw new ArgumentOutOfRangeException( "Wage",
33                    value, "Wage must be >= 0" );
34        } // end set
35    } // end property Wage
36
37    // property that gets and sets hourly employee's hours
38    public decimal Hours
39    {
40        get
41        {
42            return hours;
43        } // end get

```

```

44     set
45     {
46         if ( value >= 0 && value <= 168 ) // validation
47             hours = value;
48         else
49             throw new ArgumentOutOfRangeException( "Hours",
50                 value, "Hours must be >= 0 and <= 168" );
51     } // end set
52 } // end property Hours
53
54 // calculate earnings; override abstract method Earnings in Employee
55 public override decimal Earnings()
56 {
57     if ( Hours <= 40M ) // no overtime
58         return Wage * Hours;
59     else
60         return 40M * Wage + ( Hours - 40M ) * Wage * 1.5M;
61 } // end method Earnings
62
63 // return string representation of HourlyEmployee object
64 public override string ToString()
65 {
66     return string.Format(
67         "hourly employee: {0}\n{1}: {2:C}; {3}: {4:F}",
68         base.ToString(), "hourly wage", Wage, "hours worked", Hours );
69 } // end method ToString
70 } // end class HourlyEmployee

```

```

1 // Exercise 12.9 Solution: SalariedEmployee.cs
2 // SalariedEmployee class that extends Employee.
3 using System;
4
5 public class SalariedEmployee : Employee
6 {
7     private decimal weeklySalary;
8
9     // seven-parameter constructor
10    public SalariedEmployee( string first, string last, string ssn,
11        int month, int day, int year, decimal salary )
12        : base( first, last, ssn, month, day, year )
13    {
14        WeeklySalary = salary; // validate weekly salary via property
15    } // end seven-parameter SalariedEmployee constructor
16
17    // property that gets and sets salaried employee's weekly salary
18    public decimal WeeklySalary
19    {
20        get
21        {
22            return weeklySalary;
23        } // end get
24        set
25        {

```

```

26         if ( value >= 0 ) // validation
27             weeklySalary = value;
28         else
29             throw new ArgumentOutOfRangeException( "WeeklySalary",
30                 value, "WeeklySalary must be >= 0" );
31     } // end set
32 } // end property WeeklySalary
33
34 // calculate earnings; override abstract method Earnings in Employee
35 public override decimal Earnings()
36 {
37     return WeeklySalary;
38 } // end method Earnings
39
40 // return string representation of SalariedEmployee object
41 public override string ToString()
42 {
43     return string.Format( "salaried employee: {0}\n{1}: {2:C}",
44         base.ToString(), "weekly salary", WeeklySalary );
45 } // end method ToString
46 } // end class SalariedEmployee

```

```

1 // Exercise 12.9 Solution: PayrollSystemTest.cs
2 // Employee hierarchy test program.
3 using System;
4
5 public class PayrollSystemTest
6 {
7     public static void Main( string[] args )
8     {
9         // create derived class objects
10        SalariedEmployee salariedEmployee = new SalariedEmployee( "John",
11            "Smith", "111-11-1111", 6, 15, 1944, 800M );
12        HourlyEmployee hourlyEmployee = new HourlyEmployee( "Karen",
13            "Price", "222-22-2222", 12, 29, 1960, 16.75M, 40M );
14        CommissionEmployee commissionEmployee =
15            new CommissionEmployee( "Sue", "Jones", "333-33-3333",
16                9, 8, 1954, 10000M, .06M );
17        BasePlusCommissionEmployee basePlusCommissionEmployee =
18            new BasePlusCommissionEmployee( "Bob", "Lewis", "444-44-4444",
19                3, 2, 1965, 5000M, .04M, 300M );
20
21        Console.WriteLine( "Employees processed individually:\n" );
22
23        Console.WriteLine( "{0}\n{1}: {2:C}\n",
24            salariedEmployee, "earned", salariedEmployee.Earnings() );
25        Console.WriteLine( "{0}\n{1}: {2:C}\n",
26            hourlyEmployee, "earned", hourlyEmployee.Earnings() );
27        Console.WriteLine( "{0}\n{1}: {2:C}\n",
28            commissionEmployee, "earned", commissionEmployee.Earnings() );
29        Console.WriteLine( "{0}\n{1}: {2:C}\n",
30            basePlusCommissionEmployee,
31            "earned", basePlusCommissionEmployee.Earnings() );

```

```
32
33 // create four-element Employee array
34 Employee[] employees = new Employee[ 4 ];
35
36 // initialize array with Employees
37 employees[ 0 ] = salariedEmployee;
38 employees[ 1 ] = hourlyEmployee;
39 employees[ 2 ] = commissionEmployee;
40 employees[ 3 ] = basePlusCommissionEmployee;
41
42 int currentMonth;
43
44 // get and validate current month
45 do
46 {
47     Console.Write( "Enter the current month (1 - 12): " );
48     currentMonth = Convert.ToInt32( Console.ReadLine() );
49     Console.WriteLine();
50 } while ( ( currentMonth < 1 ) || ( currentMonth > 12 ) );
51
52 Console.WriteLine( "Employees processed polymorphically:\n" );
53
54 // generically process each element in array employees
55 foreach ( var currentEmployee in employees )
56 {
57     Console.WriteLine( currentEmployee ); // invokes ToString
58
59     // determine whether element is a BasePlusCommissionEmployee
60     if ( currentEmployee is BasePlusCommissionEmployee )
61     {
62         // downcast Employee reference to
63         // BasePlusCommissionEmployee reference
64         BasePlusCommissionEmployee employee =
65             ( BasePlusCommissionEmployee ) currentEmployee;
66
67         employee.BaseSalary *= 1.1M;
68         Console.WriteLine(
69             "new base salary with 10% increase is: {0:C}",
70             employee.BaseSalary );
71     } // end if
72
73     // if month of employee's birthday, add $100 to salary
74     if ( currentMonth == currentEmployee.BirthDate.Month )
75     {
76         Console.WriteLine(
77             "earned {0:C} {1}\n", currentEmployee.Earnings(),
78             "plus $100.00 birthday bonus" );
79     }
80     else
81     {
82         Console.WriteLine(
83             "earned {0:C}\n", currentEmployee.Earnings() );
84     }
85 } // end for
```

```

83      // get type name of each object in employees array
84      for ( int j = 0; j < employees.Length; j++ )
85          Console.WriteLine( "Employee {0} is a {1}", j,
86                             employees[ j ].GetType() );
87      } // end Main
88  } // end class PayrollSystemTest

```

```

Date object constructor for date 6/15/1944
Date object constructor for date 12/29/1960
Date object constructor for date 9/8/1954
Date object constructor for date 3/2/1965
Employees processed individually:

salaried employee: John Smith
social security number: 111-11-1111
birth date: 6/15/1944
weekly salary: $800.00
earned: $800.00

hourly employee: Karen Price
social security number: 222-22-2222
birth date: 12/29/1960
hourly wage: $16.75; hours worked: 40.00
earned: $670.00

commission employee: Sue Jones
social security number: 333-33-3333
birth date: 9/8/1954
gross sales: $10,000.00; commission rate: 0.06
earned: $600.00

base-salaried commission employee: Bob Lewis
social security number: 444-44-4444
birth date: 3/2/1965
gross sales: $5,000.00; commission rate: 0.04; base salary: $300.00
earned: $500.00

Enter the current month (1 - 12): 3
Employees processed polymorphically:

salaried employee: John Smith
social security number: 111-11-1111
birth date: 6/15/1944
weekly salary: $800.00
earned $800.00

hourly employee: Karen Price
social security number: 222-22-2222
birth date: 12/29/1960
hourly wage: $16.75; hours worked: 40.00
earned $670.00

commission employee: Sue Jones
social security number: 333-33-3333
birth date: 9/8/1954
gross sales: $10,000.00; commission rate: 0.06
earned $600.00

```

```

base-salaried commission employee: Bob Lewis
social security number: 444-44-4444
birth date: 3/2/1965
gross sales: $5,000.00; commission rate: 0.04; base salary: $300.00
new base salary with 10% increase is: $330.00
earned $530.00 plus $100.00 birthday bonus

Employee 0 is a SalariedEmployee
Employee 1 is a HourlyEmployee
Employee 2 is a CommissionEmployee
Employee 3 is a BasePlusCommissionEmployee

```

12.10 (Shape Hierarchy) Implement the Shape hierarchy shown in Fig. 11.3. You may omit the Triangle and Tetrahedron classes. Each TwoDimensionalShape should contain read-only abstract property Area to calculate the area of the two-dimensional shape. Each ThreeDimensionalShape should have read-only abstract properties Area and Volume to calculate the surface area and volume, respectively, of the three-dimensional shape. Create an application that uses an array of Shape references to objects of each concrete class in the hierarchy. The application should print a text description of the object to which each array element refers. Also, in the loop that processes all the shapes in the array, determine whether each shape is a TwoDimensionalShape or a ThreeDimensionalShape. If a shape is a TwoDimensionalShape, display its area. If a shape is a ThreeDimensionalShape, display its area and volume.

ANS:

```

1  // Exercise 12.10 Solution: Shape.cs
2  // Declaration of class Shape.
3  public abstract class Shape
4  {
5      // auto-implemented property X
6      public int X { get; set; }
7
8      // auto-implemented property Y
9      public int Y { get; set; }
10
11     // two-parameter constructor
12     public Shape( int x, int y )
13     {
14         X = x;
15         Y = y;
16     } // end two-parameter Shape constructor
17
18     // return string representation of Shape object
19     public override string ToString()
20     {
21         return string.Format( "[{0}, {1}]", X, Y );
22     }
23
24     // abstract properties
25     public abstract string Name
26     {
27         get;
28     }
29 } // end class Shape

```

```

1  // Exercise 12.10 Solution: TwoDimensionalShape.cs
2  // Declaration of class TwoDimensionalShape.
3  public abstract class TwoDimensionalShape : Shape
4  {
5      // auto-implemented property Dimension1
6      public int Dimension1 { get; set; }
7
8      // auto-implemented property Dimension2
9      public int Dimension2 { get; set; }
10
11     // four-parameter constructor
12     public TwoDimensionalShape( int x, int y, int d1, int d2 )
13         : base( x, y )
14     {
15         Dimension1 = d1;
16         Dimension2 = d2;
17     } // end four-parameter TwoDimensionalShape constructor
18
19     // abstract properties
20     public abstract double Area
21     {
22         get;
23     }
24 } // end class TwoDimensionalShape

```

```

1  // Exercise 12.10 Solution: Square.cs
2  // Declaration of class Square.
3  public class Square : TwoDimensionalShape
4  {
5      // three-parameter constructor
6      public Square( int x, int y, int side )
7          : base( x, y, side, side ) { }
8
9      // overridden properties
10     // read-only property that gets the name
11     public override string Name
12     {
13         get
14         {
15             return "Square";
16         } // end get
17     } // end method GetName
18
19     // read-only property that gets the area
20     public override double Area
21     {
22         get
23         {
24             return Side * Side;
25         } // end get
26     } // end property Area
27

```

```

28 // property that gets and sets the side
29 public int Side
30 {
31     get
32     {
33         return Dimension1;
34     } // end get
35     set
36     {
37         Dimension1 = value;
38         Dimension2 = value;
39     } // end set
40 } // end property Side
41
42 public override string ToString()
43 {
44     return string.Format( "{0} {1}: {2}\n",
45         base.ToString(), "side", Side );
46 } // end method ToString
47 } // end class Square

```

```

1 // Exercise 12.10 Solution: Circle.cs
2 // Declaration of class Circle.
3 using System;
4
5 public class Circle : TwoDimensionalShape
6 {
7     // three-parameter constructor
8     public Circle( int x, int y, int radius )
9         : base( x, y, radius, radius ) { }
10
11     // overridden properties
12     // read-only property that gets the name
13     public override string Name
14     {
15         get
16         {
17             return "Circle";
18         } // end get
19     } // end property Name
20
21     // read-only property that gets the area
22     public override double Area
23     {
24         get
25         {
26             return ( Math.PI * Radius * Radius );
27         } // end get
28     } // end property Area
29
30     // property that gets and sets the radius
31     public int Radius
32     {

```

```

33     get
34     {
35         return Dimension1;
36     } // end get
37     set
38     {
39         Dimension1 = value;
40         Dimension2 = value;
41     } // end set
42 } // end property Radius
43
44 public override string ToString()
45 {
46     return string.Format( "{0} {1}: {2}\n",
47         base.ToString(), "radius", Radius );
48 } // end method ToString
49 } // end class Circle

```

```

1 // Exercise 12.10 Solution: ThreeDimensionalShape.cs
2 // Declaration of class ThreeDimensionalShape.
3 public abstract class ThreeDimensionalShape : Shape
4 {
5     // auto-implemented property Dimension1
6     public int Dimension1 { get; set; }
7
8     // auto-implemented property Dimension2
9     public int Dimension2 { get; set; }
10
11    // auto-implemented property Dimension3
12    public int Dimension3 { get; set; }
13
14    // five-parameter constructor
15    public ThreeDimensionalShape( int x, int y, int d1,
16        int d2, int d3 )
17        : base( x, y )
18    {
19        Dimension1 = d1;
20        Dimension2 = d2;
21        Dimension3 = d3;
22    } // end five-parameter ThreeDimensionalShape constructor
23
24    // abstract properties
25    public abstract double Area
26    {
27        get;
28    }
29    public abstract double Volume
30    {
31        get;
32    }
33 } // end class ThreeDimensionalShape

```

```
1 // Exercise 12.10 Solution: Cube.cs
2 // Declaration of class Cube.
3 public class Cube : ThreeDimensionalShape
4 {
5     // three-parameter constructor
6     public Cube( int x, int y, int side )
7         : base( x, y, side, side, side ) { }
8
9     // overridden properties
10    // read-only property that gets the name
11    public override string Name
12    {
13        get
14        {
15            return "Cube";
16        } // end get
17    } // end property Name
18
19    // read-only property that gets the area
20    public override double Area
21    {
22        get
23        {
24            return ( 6 * Side * Side );
25        } // end get
26    } // end property Area
27
28    // read-only property that gets the volume
29    public override double Volume
30    {
31        get
32        {
33            return ( Side * Side * Side );
34        } // end get
35    } // end property Volume
36
37    // property that gets and sets the side length
38    public int Side
39    {
40        get
41        {
42            return Dimension1;
43        } // end get
44        set
45        {
46            Dimension1 = value;
47            Dimension2 = value;
48            Dimension3 = value;
49        } // end set
50    } // end property Side
51
52    public override string ToString()
53    {
```

```
54         return string.Format( "{0} {1}: {2}\n",
55             base.ToString(), "side", Side );
56     } // end method ToString
57 } // end class Cube
```

```
1 // Exercise 12.10 Solution: Sphere.cs
2 // Declaration of class Sphere.
3 using System;
4
5 public class Sphere : ThreeDimensionalShape
6 {
7     // three-parameter constructor
8     public Sphere( int x, int y, int radius )
9         : base( x, y, radius, radius, radius ) { }
10
11     // overridden properties
12     // read-only property that gets the name
13     public override string Name
14     {
15         get
16         {
17             return "Sphere";
18         } // end get
19     } // end property Name
20
21     // read-only property that gets the area
22     public override double Area
23     {
24         get
25         {
26             return ( 4 * Math.PI * Radius * Radius );
27         } // end get
28     } // end property Area
29
30     // read-only property that gets the volume
31     public override double Volume
32     {
33         get
34         {
35             return ( 4.0 / 3.0 * Math.PI *
36                 Radius * Radius * Radius );
37         } // end get
38     } // end property Volume
39
40     // property that gets and sets the radius
41     public int Radius
42     {
43         get
44         {
45             return Dimension1;
46         } // end get
47         set
48         {
```

```

49         Dimension1 = value;
50         Dimension2 = value;
51         Dimension3 = value;
52     } // end set
53 } // end property Radius
54
55 public override string ToString()
56 {
57     return string.Format( "{0} {1}: {2}\n",
58         base.ToString(), "radius", Radius );
59 } // end method ToString
60 } // end class Sphere

```

```

1  // Exercise 12.10 Solution: ShapeTest.cs
2  // Application tests the Shape hierarchy.
3  using System;
4
5  public class ShapeTest
6  {
7      // create Shape objects and display their information
8      public static void Main( string[] args )
9      {
10         Shape[] shapes = new Shape[ 4 ];
11         shapes[ 0 ] = new Circle( 22, 88, 4 );
12         shapes[ 1 ] = new Square( 71, 96, 10 );
13         shapes[ 2 ] = new Sphere( 8, 89, 2 );
14         shapes[ 3 ] = new Cube( 79, 61, 8 );
15
16         // call method ToString on all shapes
17         foreach ( var currentShape in shapes )
18         {
19             Console.Write( "{0}: {1}",
20                 currentShape.Name, currentShape );
21
22             if ( currentShape is TwoDimensionalShape )
23             {
24                 TwoDimensionalShape twoDimensionalShape =
25                     ( TwoDimensionalShape ) currentShape;
26
27                 Console.WriteLine( "{0}'s area is {1}",
28                     currentShape.Name, twoDimensionalShape.Area );
29             } // end if
30
31             if ( currentShape is ThreeDimensionalShape )
32             {
33                 ThreeDimensionalShape threeDimensionalShape =
34                     ( ThreeDimensionalShape ) currentShape;
35
36                 Console.WriteLine( "{0}'s area is {1}",
37                     currentShape.Name, threeDimensionalShape.Area );
38                 Console.WriteLine( "{0}'s volume is {1}",
39                     currentShape.Name,
40                     threeDimensionalShape.Volume );

```

```

41         } // end if
42
43         Console.WriteLine();
44     } // end foreach
45 } // end Main
46 } // end class ShapeTest

```

```

Circle: [22, 88] radius: 4
Circle's area is 50.2654824574367

Square: [71, 96] side: 10
Square's area is 100

Sphere: [8, 89] radius: 2
Sphere's area is 50.2654824574367
Sphere's volume is 33.5103216382911

Cube: [79, 61] side: 8
Cube's area is 384
Cube's volume is 512

```

12.11 (Payroll System Modification) Modify the payroll system of Figs. 12.4–12.9 to include an additional `Employee` derived class, `PieceWorker`, that represents an employee whose pay is based on the number of pieces of merchandise produced. Class `PieceWorker` should contain private instance variables `wage` (to store the employee's wage per piece) and `pieces` (to store the number of pieces produced). Provide a concrete implementation of method `Earnings` in class `PieceWorker` that calculates the employee's earnings by multiplying the number of pieces produced by the wage per piece. Create an array of `Employee` variables to store references to objects of each concrete class in the new `Employee` hierarchy. Display each `Employee`'s string representation and earnings.

ANS:

```

1  // Exercise 12.11 Solution: PieceWorker
2  // PieceWorker class that extends Employee.
3  using System;
4
5  public class PieceWorker : Employee
6  {
7      private decimal wage; // wage per piece
8      private int pieces; // pieces of merchandise produced in week
9
10     // five-parameter constructor
11     public PieceWorker( string first, string last, string ssn,
12         decimal wagePerPiece, int piecesProduced )
13         : base( first, last, ssn )
14     {
15         Wage = wagePerPiece; // validate wage per piece via property
16         Pieces = piecesProduced; // validate pieces produced via property
17     } // end five-parameter PieceWorker constructor
18
19     // property that gets and sets piece-worker's wage per piece
20     public decimal Wage
21     {

```

```

22     get
23     {
24         return wage;
25     } // end get
26     set
27     {
28         if ( value >= 0 )
29             wage = value;
30         else
31             throw new ArgumentOutOfRangeException( "Wage",
32                 value, "Wage must be >= 0" );
33     } // end set
34 } // end property Wage
35
36 // property that gets and sets piece-worker's number of pieces
37 public int Pieces
38 {
39     get
40     {
41         return pieces;
42     } // end get
43     set
44     {
45         if ( value >= 0 )
46             pieces = value;
47         else
48             throw new ArgumentOutOfRangeException( "Pieces",
49                 value, "Pieces must be >= 0" );
50     } // end set
51 } // end property Pieces
52
53 // calculate earnings; override abstract method Earnings in Employee
54 public override decimal Earnings()
55 {
56     return Pieces * Wage;
57 } // end method Earnings
58
59 // return string representation of PieceWorker object
60 public override string ToString()
61 {
62     return string.Format( "{0}: {1}\n{2}: {3:C}; {4}: {5}",
63         "piece worker", base.ToString(),
64         "wage per piece", Wage, "pieces produced", Pieces );
65 } // end method ToString
66 } // end class PieceWorker

```

```

1 // Exercise 12.11 Solution: PayrollSystemTest.cs
2 // Employee hierarchy test program.
3 using System;
4
5 public class PayrollSystemTest
6 {

```

```

7 public static void Main( string[] args )
8 {
9     // create five-element Employee array
10    Employee[] employees = new Employee[ 5 ];
11
12    // initialize array with Employees
13    employees[ 0 ] = new SalariedEmployee( "John", "Smith",
14        "111-11-1111", 800M );
15    employees[ 1 ] = new HourlyEmployee( "Karen", "Price",
16        "222-22-2222", 16.75M, 40M );
17    employees[ 2 ] = new CommissionEmployee( "Sue", "Jones",
18        "333-33-3333", 10000M, .06M );
19    employees[ 3 ] = new BasePlusCommissionEmployee( "Bob", "Lewis",
20        "444-44-4444", 5000M, .04M, 300M );
21    employees[ 4 ] = new PieceWorker( "Rick", "Bridges",
22        "555-55-5555", 2.25M, 400 );
23
24    Console.WriteLine( "Employees processed polymorphically:\n" );
25
26    // generically process each element in array employees
27    foreach ( var currentEmployee in employees )
28    {
29        Console.WriteLine( currentEmployee ); // invokes ToString
30        Console.WriteLine( "earned {0:C}\n",
31            currentEmployee.Earnings() );
32    } // end foreach
33 } // end Main
34 } // end class PayrollSystemTest

```

Employees processed polymorphically:

salaried employee: John Smith
social security number: 111-11-1111
weekly salary: \$800.00
earned \$800.00

hourly employee: Karen Price
social security number: 222-22-2222
hourly wage: \$16.75; hours worked: \$40.00
earned \$670.00

commission employee: Sue Jones
social security number: 333-33-3333
gross sales: \$10,000.00; commission rate: 0.06
earned \$600.00

base-salaried commission employee: Bob Lewis
social security number: 444-44-4444
gross sales: \$5,000.00; commission rate: 0.04; base salary: \$300.00
earned \$500.00

piece worker: Rick Bridges
social security number: 555-55-5555
wage per piece: \$2.25; pieces produced: 400
earned \$900.00

12.12 (Accounts Payable System Modification) In this exercise, we modify the accounts payable application of Figs. 12.11–12.15 to include the complete functionality of the payroll application of Figs. 12.4–12.9. The application should still process two Invoice objects, but now should process one object of each of the four Employee derived classes. If the object currently being processed is a BasePlusCommissionEmployee, the application should increase the BasePlusCommissionEmployee's base salary by 10%. Finally, the application should output the payment amount for each object. Complete the following steps to create the new application:

- Modify classes HourlyEmployee (Fig. 12.6) and CommissionEmployee (Fig. 12.7) to place them in the IPayable hierarchy as derived classes of the version of Employee (Fig. 12.13) that implements IPayable. [*Hint:* Change the name of method Earnings to GetPaymentAmount in each derived class.]
- Modify class BasePlusCommissionEmployee (Fig. 12.8) such that it extends the version of class CommissionEmployee created in *Part a*.
- Modify PayableInterfaceTest (Fig. 12.15) to polymorphically process two Invoices, one SalariedEmployee, one HourlyEmployee, one CommissionEmployee and one BasePlusCommissionEmployee. First, output a string representation of each IPayable object. Next, if an object is a BasePlusCommissionEmployee, increase its base salary by 10%. Finally, output the payment amount for each IPayable object.

ANS:

```

1  // Exercise 12.12 Solution: HourlyEmployee.cs
2  // HourlyEmployee class that extends Employee,
3  // which implements IPayable.
4  using System;
5
6  public class HourlyEmployee : Employee
7  {
8      private decimal wage; // wage per hour
9      private decimal hours; // hours worked for week
10
11     // five-parameter constructor
12     public HourlyEmployee( string first, string last, string ssn,
13         decimal hourlyWage, decimal hoursWorked )
14         : base( first, last, ssn )
15     {
16         Wage = hourlyWage; // validate hourly wage via property
17         Hours = hoursWorked; // validate hours worked via property
18     } // end five-parameter HourlyEmployee constructor
19
20     // property that gets and sets hourly employee's hourly wage
21     public decimal Wage
22     {
23         get
24         {
25             return wage;
26         } // end get
27         set
28         {
29             if ( value >= 0 ) // validation
30                 wage = value;

```

```

31         else
32             throw new ArgumentOutOfRangeException( "Wage",
33                 value, "Wage must be >= 0" );
34     } // end set
35 } // end property Wage
36
37 // property that gets and sets hourly employee's hours worked
38 public decimal Hours
39 {
40     get
41     {
42         return hours;
43     } // end get
44     set
45     {
46         if ( value >= 0 && value <= 168 ) // validation
47             hours = value;
48         else
49             throw new ArgumentOutOfRangeException( "Hours",
50                 value, "Hours must be >= 0 and <= 168" );
51     } // end set
52 } // end property Hours
53
54 // calculate earnings; implement interface IPayable method not
55 // implemented by base class Employee
56 public override decimal GetPaymentAmount()
57 {
58     if ( Hours <= 40M ) // no overtime
59         return Wage * Hours;
60     else
61         return 40M * Wage + ( Hours - 40M ) * Wage * 1.5M;
62 } // end method GetPaymentAmount
63
64 // return string representation of HourlyEmployee object
65 public override string ToString()
66 {
67     return string.Format(
68         "hourly employee: {0}\n{1}: {2:C}; {3}: {4:F}",
69         base.ToString(), "hourly wage", Wage,
70         "hours worked", Hours );
71 } // end method ToString
72 } // end class HourlyEmployee

```

```

1 // Exercise 12.12 Solution: CommissionEmployee.cs
2 // CommissionEmployee class that extends Employee,
3 // which implements IPayable.
4 using System;
5
6 public class CommissionEmployee : Employee
7 {
8     private decimal grossSales; // gross weekly sales
9     private decimal commissionRate; // commission percentage
10

```

```
11 // five-parameter constructor
12 public CommissionEmployee( string first, string last, string ssn,
13     decimal sales, decimal rate )
14     : base( first, last, ssn )
15 {
16     GrossSales = sales; // validate gross sales via property
17     CommissionRate = rate; // validate commission rate via property
18 } // end five-parameter CommissionEmployee constructor
19
20 // property that gets and sets commission employee's gross sales
21 public decimal GrossSales
22 {
23     get
24     {
25         return grossSales;
26     } // end get
27     set
28     {
29         if ( value >= 0 )
30             grossSales = value;
31         else
32             throw new ArgumentOutOfRangeException(
33                 "GrossSales", value, "GrossSales must be >= 0" );
34     } // end set
35 } // end property GrossSales
36
37 // property that gets and sets commission employee's commission rate
38 public decimal CommissionRate
39 {
40     get
41     {
42         return commissionRate;
43     } // end get
44     set
45     {
46         if ( value > 0 && value < 1 )
47             commissionRate = value;
48         else
49             throw new ArgumentOutOfRangeException( "CommissionRate",
50                 value, "CommissionRate must be > 0 and < 1" );
51     } // end set
52 } // end property CommissionRate
53
54 // calculate earnings; implement interface IPayable method not
55 // implemented by base class Employee
56 public override decimal GetPaymentAmount()
57 {
58     return CommissionRate * GrossSales;
59 } // end method GetPaymentAmount
60
61 // return string representation of CommissionEmployee object
62 public override string ToString()
63 {
```

```

64         return string.Format( "{0}: {1}\n{2}: {3:C}; {4}: {5:F}",
65             "commission employee", base.ToString(),
66             "gross sales", GrossSales,
67             "commission rate", CommissionRate );
68     } // end method ToString
69 } // end class CommissionEmployee

```

```

1  // Exercise 12.12 Solution: BasePlusCommissionEmployee.cs
2  // BasePlusCommissionEmployee class that extends CommissionEmployee.
3  using System;
4
5  public class BasePlusCommissionEmployee : CommissionEmployee
6  {
7      private decimal baseSalary; // base salary per week
8
9      // six-parameter constructor
10     public BasePlusCommissionEmployee( string first, string last,
11         string ssn, decimal sales, decimal rate, decimal salary )
12         : base( first, last, ssn, sales, rate )
13     {
14         BaseSalary = salary; // validate and store base salary
15     } // end six-parameter BasePlusCommissionEmployee constructor
16
17     // property that gets and sets
18     // base-salaried commission employee's base salary
19     public decimal BaseSalary
20     {
21         get
22         {
23             return baseSalary;
24         } // end get
25         set
26         {
27             if ( value >= 0 )
28                 baseSalary = value;
29             else
30                 throw new ArgumentOutOfRangeException( "BaseSalary",
31                     value, "BaseSalary must be >= 0" );
32         } // end set
33     } // end property BaseSalary
34
35     // calculate earnings; override CommissionEmployee implementation of
36     // interface IPayable method
37     public override decimal GetPaymentAmount()
38     {
39         return BaseSalary + base.GetPaymentAmount();
40     } // end method GetPaymentAmount
41
42     // return string representation of BasePlusCommissionEmployee object
43     public override string ToString()
44     {

```

```

45     return string.Format( "{0} {1}; {2}: {3:C}",
46         "base-salaried", base.ToString(),
47         "base salary", BaseSalary );
48     } // end method ToString
49 } // end class BasePlusCommissionEmployee

```

```

1  // Exercise 12.12 Solution: PayableInterfaceTest.cs
2  // Tests interface IPayable.
3  using System;
4
5  public class PayableInterfaceTest
6  {
7      public static void Main( string[] args )
8      {
9          // create six-element IPayable array
10         IPayable[] payableObjects = new IPayable[ 6 ];
11
12         // populate array with objects that implement IPayable
13         payableObjects[ 0 ] = new Invoice( "01234", "seat", 2, 375M );
14         payableObjects[ 1 ] = new Invoice( "56789", "tire", 4, 79.95M );
15         payableObjects[ 2 ] = new SalariedEmployee( "John", "Smith",
16             "111-11-1111", 800M );
17         payableObjects[ 3 ] = new HourlyEmployee( "Karen", "Price",
18             "222-22-2222", 16.75M, 40M );
19         payableObjects[ 4 ] = new CommissionEmployee( "Sue", "Jones",
20             "333-33-3333", 10000M, .06M );
21         payableObjects[ 5 ] = new BasePlusCommissionEmployee( "Bob",
22             "Lewis", "444-44-4444", 5000M, .04M, 300M );
23
24         Console.WriteLine(
25             "Invoices and Employees processed polymorphically:\n" );
26
27         // generically process each element in array payableObjects
28         foreach ( var currentPayable in payableObjects )
29         {
30             // output currentPayable and its appropriate payment amount
31             Console.WriteLine( "{0}", currentPayable.ToString() );
32
33             if ( currentPayable is BasePlusCommissionEmployee )
34             {
35                 // downcast IPayable reference to
36                 // BasePlusCommissionEmployee reference
37                 BasePlusCommissionEmployee employee =
38                     ( BasePlusCommissionEmployee ) currentPayable;
39
40                 employee.BaseSalary *= 1.1M;
41                 Console.WriteLine(
42                     "new base salary with 10% increase is: {0:C}",
43                     employee.BaseSalary );
44             } // end if
45         }

```

```

46         Console.WriteLine( "{0}: {1:C}\n",
47             "payment due", currentPayable.GetPaymentAmount() );
48     } // end foreach
49 } // end Main
50 } // end class PayableInterfaceTest

```

Invoices and Employees processed polymorphically:

```

invoice:
part number: 01234 (seat)
quantity: 2
price per item: $375.00
payment due: $750.00

invoice:
part number: 56789 (tire)
quantity: 4
price per item: $79.95
payment due: $319.80

salaried employee: John Smith
social security number: 111-11-1111
weekly salary: $800.00
payment due: $800.00

hourly employee: Karen Price
social security number: 222-22-2222
hourly wage: $16.75; hours worked: 40.00
payment due: $670.00

commission employee: Sue Jones
social security number: 333-33-3333
gross sales: $10,000.00; commission rate: 0.06
payment due: $600.00

base-salaried commission employee: Bob Lewis
social security number: 444-44-4444
gross sales: $5,000.00; commission rate: 0.04; base salary: $300.00
new base salary with 10% increase is: $330.00
payment due: $530.00

```

12.13 (Package Inheritance Hierarchy) Use the Package inheritance hierarchy created in Exercise 11.8 to create an application that displays the address information and calculates the shipping costs for several Packages. The application should contain an array of Package objects of classes TwoDayPackage and OvernightPackage. Loop through the array to process the Packages polymorphically. For each Package, use properties to obtain the address information of the sender and the recipient, then print the two addresses as they would appear on mailing labels. Also, call each Package's CalculateCost method and print the result. Keep track of the total shipping cost for all Packages in the array, and display this total when the loop terminates.

ANS:

```

1 // Exercise 12.13 Solution: Package.cs
2 // Package class declaration.
3 using System;
4
5 class Package
6 {
7     private decimal weight; // weight of the package
8     private decimal costPerOunce; // cost per ounce to ship the package
9
10    // auto-implemented property SenderName
11    public string SenderName { get; set; }
12
13    // auto-implemented property SenderAddress
14    public string SenderAddress { get; set; }
15
16    // auto-implemented property SenderCity
17    public string SenderCity { get; set; }
18
19    // auto-implemented property SenderState
20    public string SenderState { get; set; }
21
22    // auto-implemented property SenderZIP
23    public string SenderZIP { get; set; }
24
25    // auto-implemented property RecipientName
26    public string RecipientName { get; set; }
27
28    // auto-implemented property RecipientAddress
29    public string RecipientAddress { get; set; }
30
31    // auto-implemented property RecipientCity
32    public string RecipientCity { get; set; }
33
34    // auto-implemented property RecipientState
35    public string RecipientState { get; set; }
36
37    // auto-implemented property RecipientZIP
38    public string RecipientZIP { get; set; }
39
40    // constructor initializes instance variables and properties
41    public Package( string senderName, string senderAddress,
42                  string senderCity, string senderState, string senderZIP,
43                  string recipientName, string recipientAddress,
44                  string recipientCity, string recipientState,
45                  string recipientZIP, decimal packageWeight, decimal cost )
46    {
47        SenderName = senderName;
48        SenderAddress = senderAddress;
49        SenderCity = senderCity;
50        SenderState = senderState;
51        SenderZIP = senderZIP;
52        RecipientName = recipientName;

```

```

53     RecipientAddress = recipientAddress;
54     RecipientCity = recipientCity;
55     RecipientState = recipientState;
56     RecipientZIP = recipientZIP;
57     Weight = packageWeight;
58     CostPerOunce = cost;
59 } // end Package constructor
60
61 // property that gets and sets package weight
62 public decimal Weight
63 {
64     get
65     {
66         return weight;
67     } // end get
68     set
69     {
70         if ( value >= 0M )
71             weight = value;
72         else
73             throw new ArgumentOutOfRangeException( "Weight",
74                 value, "Weight must be >= 0" );
75     } // end set
76 } // end property Weight
77
78 // property that gets and sets cost per ounce to ship package
79 public decimal CostPerOunce
80 {
81     get
82     {
83         return costPerOunce;
84     } // end get
85     set
86     {
87         if ( value >= 0M )
88             costPerOunce = value;
89         else
90             throw new ArgumentOutOfRangeException( "CostPerOunce",
91                 value, "CostPerOunce must be >= 0" );
92     } // end set
93 } // end property CostPerOunce
94
95 // calculate shipping cost for package
96 public virtual decimal CalculateCost()
97 {
98     return Weight * CostPerOunce;
99 } // end method CalculateCost
100 } // end class Package

```

```

1 // Exercise 12.13 Solution: OvernightPackage.cs
2 // OvernightPackage class declaration that extends Package.
3 using System;
4

```

```

5  class OvernightPackage : Package
6  {
7      private decimal overnightFeePerOunce; // overnight delivery fee/ounce
8
9      // constructor
10     public OvernightPackage( string senderName,
11         string senderAddress, string senderCity, string senderState,
12         string senderZIP, string recipientName, string recipientAddress,
13         string recipientCity, string recipientState, string recipientZIP,
14         decimal weight, decimal cost, decimal fee )
15         : base( senderName, senderAddress, senderCity, senderState,
16             senderZIP, recipientName, recipientAddress, recipientCity,
17             recipientState, recipientZIP, weight, cost )
18     {
19         OvernightFeePerOunce = fee; // set overnight fee via property
20     } // end OvernightPackage constructor
21
22     // property that gets and sets the overnight fee per ounce
23     public decimal OvernightFeePerOunce
24     {
25         get
26         {
27             return overnightFeePerOunce;
28         } // end get
29         set
30         {
31             if ( value >= 0M )
32                 overnightFeePerOunce = value;
33             else
34                 throw new ArgumentOutOfRangeException( "OvernightFeePerOunce",
35                     value, "OvernightFeePerOunce must be >= 0" );
36         } // end set
37     } // end property OvernightFeePerOunce
38
39     // calculate shipping cost for package
40     public override decimal CalculateCost()
41     {
42         return Weight * ( CostPerOunce + OvernightFeePerOunce );
43     } // end method CalculateCost
44 } // end class OvernightPackage

```

```

1  // Exercise 12.13 Solution: TwoDayPackage.cs
2  // TwoDayPackage class declaration that extends Package.
3  using System;
4
5  class TwoDayPackage : Package
6  {
7      private decimal flatFee; // flat fee for two-day-delivery service
8

```

```

9      // constructor
10     public TwoDayPackage( string senderName,
11        string senderAddress, string senderCity, string senderState,
12        string senderZIP, string recipientName, string recipientAddress,
13        string recipientCity, string recipientState, string recipientZIP,
14        decimal weight, decimal cost, decimal fee )
15        : base( senderName, senderAddress, senderCity, senderState,
16        senderZIP, recipientName, recipientAddress, recipientCity,
17        recipientState, recipientZIP, weight, cost )
18    {
19        FlatFee = fee; // validate flat fee via property
20    } // end TwoDayPackage constructor
21
22    // property that gets and sets two-day package's flat fee
23    public decimal FlatFee
24    {
25        get
26        {
27            return flatFee;
28        } // end get
29        set
30        {
31            if ( value >= 0M )
32                flatFee = value;
33            else
34                throw new ArgumentOutOfRangeException( "FlatFee",
35                value, "FlatFee must be >= 0" );
36        } // end set
37    } // end property FlatFee
38
39    // calculate shipping cost for package
40    public override decimal CalculateCost()
41    {
42        return base.CalculateCost() + FlatFee;
43    } // end method CalculateCost
44 } // end class TwoDayPackage

```

```

1  // Exercise 12.13 Solution: PackageTest.cs
2  // Processing Packages polymorphically.
3  using System;
4
5  public class PackageTest
6  {
7      public static void Main( string[] args )
8      {
9          // create packages array
10         Package[] packages = new Package[ 3 ];
11
12         // initialize array with Packages
13         packages[ 0 ] = new Package( "Lou Brown", "1 Main St",
14             "Boston", "MA", "11111", "Mary Smith", "7 Elm St",
15             "New York", "NY", "22222", 8.5M, .5M );

```

```

16 packages[ 1 ] = new TwoDayPackage( "Lisa Klein", "5 Broadway",
17   "Somerville", "MA", "33333", "Bob George", "21 Pine Rd",
18   "Cambridge", "MA", "44444", 10.5M, .65M, 2M );
19 packages[ 2 ] = new OvernightPackage( "Ed Lewis", "2 Oak St",
20   "Boston", "MA", "55555", "Don Kelly", "9 Main St",
21   "Denver", "CO", "66666", 12.25M, .7M, .25M );
22
23 decimal totalShippingCost = 0M;
24
25 // display each package's information and cost
26 for ( int i = 0; i < packages.Length; i++ )
27 {
28     Console.WriteLine( "Package {0}:\n", i + 1 );
29     Console.WriteLine( "Sender:\n{0}\n{1}\n{2}, {3} {4}\n",
30       packages[ i ].SenderName, packages[ i ].SenderAddress,
31       packages[ i ].SenderCity, packages[ i ].SenderState,
32       packages[ i ].SenderZIP );
33     Console.WriteLine( "Recipient:\n{0}\n{1}\n{2}, {3} {4}\n",
34       packages[ i ].RecipientName, packages[ i ].RecipientAddress,
35       packages[ i ].RecipientCity, packages[ i ].RecipientState,
36       packages[ i ].RecipientZIP );
37
38     decimal cost = packages[ i ].CalculateCost();
39     Console.Write( "Cost: {0:C}\n\n", cost );
40
41     // add this Package's cost to total
42     totalShippingCost += cost;
43 } // end for
44
45 Console.WriteLine( "\nTotal shipping cost: {0:C}",
46   totalShippingCost );
47 } // end Main
48 } // end class PackageTest

```

Package 1:

Sender:
 Lou Brown
 1 Main St
 Boston, MA 11111

Recipient:
 Mary Smith
 7 Elm St
 New York, NY 22222

Cost: \$4.25

Package 2:

Sender:

Lisa Klein
5 Broadway
Somerville, MA 33333

Recipient:

Bob George
21 Pine Rd
Cambridge, MA 44444

Cost: \$8.83

Package 3:

Sender:

Ed Lewis
2 Oak St
Boston, MA 55555

Recipient:

Don Kelly
9 Main St
Denver, CO 66666

Cost: \$11.64

Total shipping cost: \$24.71

12.14 (*Polymorphic Banking Program Using Account Hierarchy*) Develop a polymorphic banking application using the Account hierarchy created in Exercise 11.9. Create an array of Account references to SavingsAccount and CheckingAccount objects. For each Account in the array, allow the user to specify an amount of money to withdraw from the Account using method Debit and an amount of money to deposit into the Account using method Credit. As you process each Account, determine its type. If an Account is a SavingsAccount, calculate the amount of interest owed to the Account using method CalculateInterest, then add the interest to the account balance using method Credit. After processing an Account, print the updated account balance obtained by using base-class property Balance.

ANS:

```

1 // Exercise 12.14 Solution: Account.cs
2 // Account class declaration.
3 using System;
4
5 class Account
6 {
7     private decimal balance; // stores the account balance
8
9     // Account constructor initializes instance variable balance
10    public Account( decimal initialBalance )
11    {

```

```

12     Balance = initialBalance; // validate balance via property
13 } // end Account constructor
14
15 // credit the account balance by amount
16 public virtual void Credit( decimal amount )
17 {
18     Balance += amount; // add amount to balance
19 } // end method Credit
20
21 // debit the account balance by amount;
22 // return bool indicating whether debit was successful
23 public virtual bool Debit( decimal amount )
24 {
25     if ( amount > Balance ) // debit amount exceeds balance
26     {
27         Console.WriteLine( "Debit amount exceeded account balance." );
28         return false;
29     } // end if
30     else // debit amount does not exceed balance
31     {
32         Balance -= amount;
33         return true;
34     } // end else
35 } // end method Debit
36
37 // property that gets and sets the account balance
38 public decimal Balance
39 {
40     get
41     {
42         return balance;
43     } // end get
44     set
45     {
46         // if value is greater than or equal to 0,
47         // set value as the balance of the Account
48         if ( value >= 0M )
49             balance = value;
50         else
51             throw new ArgumentOutOfRangeException( "Balance",
52             value, "Balance must be >= 0" );
53     } // end set
54 } // end property Balance
55 } // end class Account

```

```

1 // Exercise 12.14 Solution: CheckingAccount.cs
2 // CheckingAccount class declaration.
3 using System;
4
5 class CheckingAccount : Account
6 {
7     private decimal transactionFee; // fee charged per transaction
8

```

```

 9 // constructor initializes balance and transaction fee
10 public CheckingAccount( decimal initialBalance, decimal fee )
11     : base( initialBalance ) // initialize base class members
12 {
13     if ( fee >= 0M )
14         transactionFee = fee;
15     else
16         throw new ArgumentOutOfRangeException( "fee",
17             fee, "fee must be >= 0" );
18 } // end CheckingAccount constructor
19
20 // credit the account balance by amount and charge fee
21 public override void Credit( decimal amount )
22 {
23     base.Credit( amount ); // always succeeds
24     ChargeFee();
25 } // end method Credit
26
27 // debit the account balance by amount and charge fee
28 public override bool Debit( decimal amount )
29 {
30     bool success = base.Debit( amount ); // attempt to debit
31
32     if ( success ) // if money was debited, charge fee and return true
33     {
34         ChargeFee();
35         return true;
36     } // end if
37     else // otherwise, do not charge fee and return false
38         return false;
39 } // end method Debit
40
41 // subtract transaction fee
42 private void ChargeFee()
43 {
44     Balance = Balance - transactionFee;
45     Console.WriteLine( "{0:C} transaction fee charged.",
46         transactionFee );
47 } // end method ChargeFee
48 } // end class CheckingAccount

```

```

1 // Exercise 12.14 Solution: SavingsAccount.cs
2 // SavingsAccount class declaration.
3 class SavingsAccount : Account
4 {
5     // interest rate (percentage) earned by account
6     private decimal interestRate;
7
8     // constructor initializes balance and interest rate
9     public SavingsAccount( decimal initialBalance, decimal rate )
10         : base( initialBalance ) // initialize base class member
11     {

```

```

12     if ( rate >= 0M )
13         interestRate = rate;
14     else
15         throw new ArgumentOutOfRangeException( "rate",
16             rate, "rate must be >= 0" );
17 } // end SavingsAccount constructor
18
19 // return the amount of interest earned
20 public decimal CalculateInterest()
21 {
22     return Balance * interestRate;
23 } // end method CalculateInterest
24 } // end class SavingsAccount

```

```

1 // Exercise 12.14 Solution: AccountTest.cs
2 // Processing Accounts polymorphically.
3 using System;
4
5 public class AccountTest
6 {
7     public static void Main( string[] args )
8     {
9         // create array of accounts
10        Account[] accounts = new Account[ 4 ];
11
12        // initialize array with Accounts
13        accounts[ 0 ] = new SavingsAccount( 25M, .03M );
14        accounts[ 1 ] = new CheckingAccount( 80M, 1M );
15        accounts[ 2 ] = new SavingsAccount( 200M, .015M );
16        accounts[ 3 ] = new CheckingAccount( 400M, .5M );
17
18        // loop through array, prompting user for debit and credit amounts
19        for ( int i = 0; i < accounts.Length; i++ )
20        {
21            Console.WriteLine( "Account {0} balance: {1:C}",
22                i + 1, accounts[ i ].Balance );
23
24            Console.Write(
25                "\nEnter an amount to withdraw from Account {0}: ", i + 1 );
26            decimal withdrawalAmount =
27                Convert.ToDecimal( Console.ReadLine() );
28
29            accounts[ i ].Debit( withdrawalAmount ); // attempt to debit
30
31            Console.Write(
32                "\nEnter an amount to deposit into Account {0}: ", i + 1 );
33            decimal depositAmount =
34                Convert.ToDecimal( Console.ReadLine() );
35
36            // credit amount to Account
37            accounts[ i ].Credit( depositAmount );
38

```

```

39         // if Account is a SavingsAccount, calculate and add interest
40         if ( accounts[ i ] is SavingsAccount )
41         {
42             // downcast
43             SavingsAccount currentAccount =
44                 ( SavingsAccount ) accounts[ i ];
45
46             decimal interestEarned = currentAccount.CalculateInterest();
47             Console.WriteLine(
48                 "Adding {0:C} interest to Account {1} (a SavingsAccount)",
49                 interestEarned, i + 1 );
50             currentAccount.Credit( interestEarned );
51         } // end if
52
53         Console.WriteLine( "\nUpdated Account {0} balance: {1:C}\n\n",
54                             i + 1, accounts[ i ].Balance );
55     } // end for
56 } // end Main
57 } // end class AccountTest

```

Account 1 balance: \$25.00

Enter an amount to withdraw from Account 1: 10

Enter an amount to deposit into Account 1: 20

Adding \$1.05 interest to Account 1 (a SavingsAccount)

Updated Account 1 balance: \$36.05

Account 2 balance: \$80.00

Enter an amount to withdraw from Account 2: 50

\$1.00 transaction fee charged.

Enter an amount to deposit into Account 2: 5

\$1.00 transaction fee charged.

Updated Account 2 balance: \$33.00

Account 3 balance: \$200.00

Enter an amount to withdraw from Account 3: 1000

Debit amount exceeded account balance.

Enter an amount to deposit into Account 3: 5

Adding \$3.08 interest to Account 3 (a SavingsAccount)

Updated Account 3 balance: \$208.08

Account 4 balance: \$400.00

Enter an amount to withdraw from Account 4: 100

\$0.50 transaction fee charged.

Enter an amount to deposit into Account 4: 200

\$0.50 transaction fee charged.

Updated Account 4 balance: \$499.00

Making a Difference Exercise

12.15 (*CarbonFootprint Interface: Polymorphism*) Using interfaces, as you learned in this chapter, you can specify similar behaviors for possibly disparate classes. Governments and companies worldwide are becoming increasingly concerned with carbon footprints (annual releases of carbon dioxide into the atmosphere) from buildings burning various types of fuels for heat, vehicles burning fuels for power, and the like. Many scientists blame these greenhouse gases for the phenomenon called global warming. Create three small classes unrelated by inheritance—classes `Building`, `Car` and `Bicycle`. Give each class some unique appropriate attributes and behaviors that it does not have in common with other classes. Write an interface `CarbonFootprint` with a `getCarbonFootprint` method. Have each of your classes implement that interface, so that its `getCarbonFootprint` method calculates an appropriate carbon footprint for that class (check out a few websites that explain how to calculate carbon footprints). Write an application that creates objects of each of the three classes, places references to those objects in `ArrayList<CarbonFootprint>`, then iterates through the `ArrayList`, polymorphically invoking each object's `getCarbonFootprint` method. For each object, print some identifying information and the object's carbon footprint.

ANS: See solution code in source files.

13

Exception Handling: A Deeper Look: Solutions

It is common sense to take a method and try it. If it fails, admit it frankly and try another. But above all, try something.

—Franklin Delano Roosevelt

*O! throw away the
worse part of it,
And live the purer
with the other half.*

—William Shakespeare

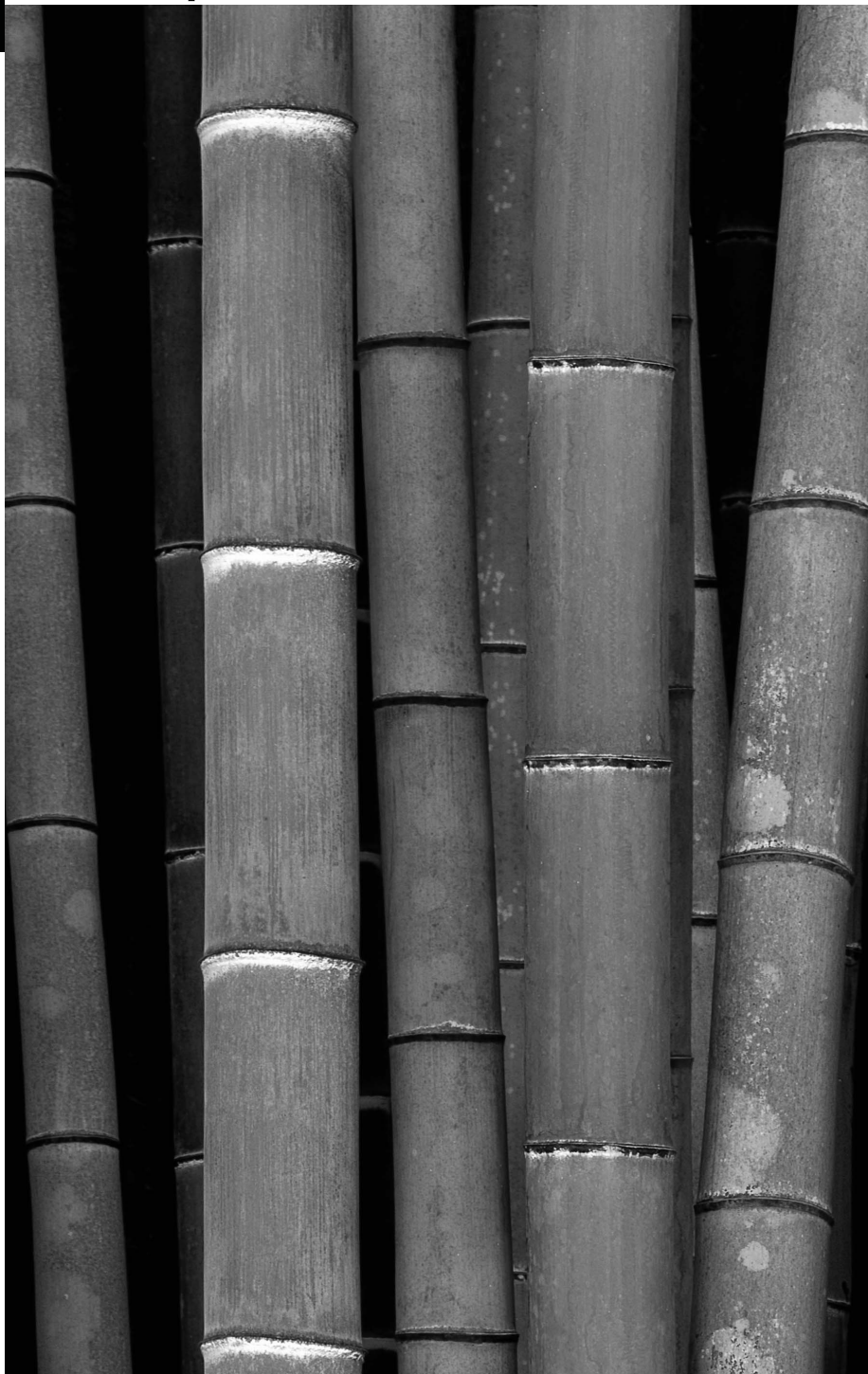
*If they're running and they don't
look where they're going I have
to come out from somewhere
and catch them.*

—J. D. Salinger

Objectives

In this chapter you'll learn:

- What exceptions are and how they're handled.
- When to use exception handling.
- To use `try` blocks to delimit code in which exceptions might occur.
- To `throw` exceptions to indicate a problem.
- To use `catch` blocks to specify exception handlers.
- To use the `finally` block to release resources.
- The .NET exception class hierarchy.
- `Exception` properties.
- To create user-defined exceptions.



Self-Review Exercises

13.1 Fill in the blanks in each of the following statements:

a) A method is said to _____ an exception when that method detects that a problem has occurred.

ANS: throw.

b) When present, the _____ block associated with a try block always executes.

ANS: finally.

c) Exception classes are derived from class _____.

ANS: Exception.

d) The statement that throws an exception is called the _____ of the exception.

ANS: throw point.

e) C# uses the _____ model of exception handling as opposed to the _____ model of exception handling.

ANS: termination, resumption.

f) An uncaught exception in a method causes the method to _____ from the method-call stack.

ANS: unwind.

g) Method `Convert.ToInt32` can throw a(n) _____ exception if its argument is not a valid integer value.

ANS: `FormatException`.

13.2 State whether each of the following is *true* or *false*. If *false*, explain why.

a) Exceptions always are handled in the method that initially detects the exception.

ANS: False. Exceptions can be handled by other methods on the method-call stack.

b) User-defined exception classes should extend class `SystemException`.

ANS: False. User-defined exception classes should typically extend class `Exception`.

c) Accessing an out-of-bounds array index causes the CLR to throw an exception.

ANS: True.

d) A `finally` block is optional after a `try` block that does not have any corresponding catch blocks.

ANS: False. A `try` block that does not have any catch blocks requires a `finally` block.

e) A `finally` block is guaranteed to execute.

ANS: False. The `finally` block executes only if program control enters the corresponding `try` block.

f) It is possible to return to the throw point of an exception using keyword `return`.

ANS: False. `return` causes control to return to the caller.

g) Exceptions can be rethrown.

ANS: True.

h) Property `Message` of class `Exception` returns a string indicating the method from which the exception was thrown.

ANS: False. Property `Message` of class `Exception` returns a string representing the error message.

Exercises

13.3 Use inheritance to create an exception base class and various exception-derived classes. Write a program to demonstrate that the catch specifying the base class catches derived-class exceptions.

ANS: [Note: Since this is a mechanical exercise, we did not provide the complete set of constructors for each exception class.]

```
1 // Exercise 13.3 Solution: ExceptionTest.cs
2 // Demonstrates catching derived-class exceptions.
3 using System;
4
5 // create System.Exception derived class
6 class ExceptionA : Exception
7 {
8     // empty body
9 } // end class ExceptionA
10
11 // create class ExceptionA derived class
12 class ExceptionB : ExceptionA
13 {
14     // empty body
15 } // end class ExceptionB
16
17 // create class ExceptionB derived class
18 class ExceptionC : ExceptionB
19 {
20     // empty body
21 } // end class ExceptionC
22
23 // ExceptionTest throws and catches derived-class exceptions
24 public class ExceptionTest
25 {
26     public static void Main( string[] args )
27     {
28         // throw and catch an ExceptionC exception
29         try
30         {
31             // throw derived-class ExceptionC
32             throw new ExceptionC();
33         } // end try
34         catch ( ExceptionA exceptionTest )
35         {
36             Console.WriteLine( "ExceptionC caught in ExceptionA " +
37                               "catch block: \n" + exceptionTest + "\n" );
38         } // end catch
39
40         // throw and catch an ExceptionB exception
41         try
42         {
43             // throw derived-class ExceptionB
44             throw new ExceptionB();
45         } // end try
46         catch ( ExceptionA exceptionTest )
47         {
48             Console.WriteLine( "ExceptionB caught in ExceptionA " +
49                               "catch block: \n" + exceptionTest + "\n" );
50         } // end catch
51     } // end Main
52 } // end class ExceptionTest
```

```
ExceptionC caught in ExceptionA catch block:
ExceptionC: Exception of type 'ExceptionC' was thrown.
    at ExceptionTest.Main(String[] args) in C:\solutions\sol_Ch13\Ex13_03\Ex-
ceptionTest\ExceptionTest.cs:line 32
```

```
ExceptionB caught in ExceptionA catch block:
ExceptionB: Exception of type 'ExceptionB' was thrown.
    at ExceptionTest.Main(String[] args) in C:\solutions\sol_Ch13\Ex13_03\Ex-
ceptionTest\ExceptionTest.cs:line 44
```

13.4 Write a program that demonstrates how various exceptions are caught with

catch (Exception exceptionParameter)

ANS: [Note: Since this is a mechanical exercise, we did not provide the complete set of constructors for each exception class.]

```
1 // Exercise 13.4 Solution: ExceptionTest.cs
2 // Catches various exceptions with class Exception.
3 using System;
4
5 // create System.Exception derived class
6 class ExceptionA : Exception
7 {
8     // empty body
9 } // end class ExceptionA
10
11 // create class ExceptionA derived class
12 class ExceptionB : ExceptionA
13 {
14     // empty body
15 } // end class ExceptionB
16
17 // ExceptionTest use System.Exception to handle exceptions
18 public class ExceptionTest
19 {
20     public static void Main( string[] args )
21     {
22         // throw and catch an ExceptionA exception
23         try
24         {
25             throw new ExceptionA();
26         } // end try
27         catch ( Exception exception1 )
28         {
29             Console.WriteLine( exception1 + "\n" );
30         } // end catch
31
32         // throw and catch an ExceptionB exception
33         try
34         {
```

```

35         throw new ExceptionB();
36     } // end try
37     catch ( Exception exception2 )
38     {
39         Console.WriteLine( exception2 + "\n" );
40     } // end catch
41
42     // throw and catch a NullReferenceException
43     try
44     {
45         throw new NullReferenceException();
46     } // end try
47     catch ( Exception exception3 )
48     {
49         Console.WriteLine( exception3 + "\n" );
50     } // end catch
51 } // end Main
52 } // end class ExceptionTest

```

```

ExceptionA: Exception of type 'ExceptionA' was thrown.
    at ExceptionTest.Main(String[] args) in C:\solutions\sol_Ch13\Ex13_04\
    ExceptionTest\ExceptionTest.cs:line 25

ExceptionB: Exception of type 'ExceptionB' was thrown.
    at ExceptionTest.Main(String[] args) in C:\solutions\sol_Ch13\Ex13_04\
    ExceptionTest\ExceptionTest.cs:line 35

System.NullReferenceException: Object reference not set to an instance of an
object.
    at ExceptionTest.Main(String[] args) in C:\solutions\sol_Ch13\Ex13_04\
    ExceptionTest\ExceptionTest.cs:line 45

```

13.5 To demonstrate the importance of the order of exception handlers, write two programs, one with correct ordering of catch blocks (i.e., place the base-class exception handler after all derived-class exception handlers) and another with improper ordering (i.e., place the base-class exception handler before the derived-class exception handlers). What happens when you attempt to compile the second program.

ANS: The second program generates errors, indicating the improper order of catch blocks.

```

1  // Exercise 13.5a Solution: OrderTest.cs
2  // Show logic for proper ordering of catch statements.
3  using System;
4
5  public class OrderTest
6  {
7      public static void Main( string[] args )
8      {
9          // throw three derived-class exceptions; each exception
10         // is handled by specific derived-class exception handler
11         for ( int i = 0; i < 3; i++ )
12         {
13             try
14             {

```

```

15         if ( i == 0 )
16         {
17             throw new System.IO.IOException(); // throw an IOException
18         } // end if
19         else if ( i == 1 )
20         {
21             // throw a FormatException
22             throw new FormatException();
23         } // end else if
24         else if ( i == 2 )
25         {
26             // throw an ArithmeticException
27             throw new ArithmeticException();
28         } // end else if
29     } // end try
30     catch ( System.IO.IOException ioError )
31     {
32         // exception handler catches IOExceptions
33         Console.WriteLine( ioError.Message );
34         Console.WriteLine(
35             "Perform logic for handling IOException\n" );
36
37         // perform logic for handling IOException
38     } // end catch
39     catch ( FormatException formatError )
40     {
41         // exception handler catches FormatExceptions
42         Console.WriteLine( formatError.Message );
43         Console.WriteLine( "Perform logic for handling " +
44             "FormatException\n" );
45
46         // perform logic for handling FormatException
47     } // end catch
48     catch ( ArithmeticException arithmeticError )
49     {
50         // exception handler catches ArithmeticException
51         Console.WriteLine( arithmeticError.Message );
52         Console.WriteLine( "Perform logic for handling " +
53             "ArithmeticException\n" );
54
55         // perform logic for handling ArithmeticException
56     } // end catch
57     catch ( Exception baseClassException )
58     {
59         // catch-all exception handler
60         Console.WriteLine( "Base-class exception handler invoked: " +
61             baseClassException.Message );
62     } // end catch
63 } // end for
64 } // end Main
65 } // end class OrderTest

```

I/O error occurred.
Perform logic for handling IOException

One of the identified items was in an invalid format.
Perform logic for handling FormatException

Overflow or underflow in the arithmetic operation.
Perform logic for handling ArithmeticException

```

1  // Exercise 13.5b Solution: OrderTest.cs
2  // Show compilation errors due to improper ordering of catch blocks.
3  using System;
4
5  public class OrderTest
6  {
7      public static void Main( string[] args )
8      {
9          // throw three derived-class exceptions; each exception
10         // is handled by base-class exception handler
11         for ( int i = 0; i < 3; i++ )
12         {
13             try
14             {
15                 if ( i == 0 )
16                 {
17                     throw new System.IO.IOException(); // throw an IOException
18                 } // end if
19                 else if ( i == 1 )
20                 {
21                     // throw a FormatException
22                     throw new FormatException();
23                 } // end else if
24                 else if ( i == 2 )
25                 {
26                     // throw an ArithmeticException
27                     throw new ArithmeticException();
28                 } // end else if
29             } // end try
30             catch ( Exception baseClassException )
31             {
32                 // catch exception of any type
33                 Console.WriteLine( "Base-class exception handler invoked: " +
34                     baseClassException.Message );
35             } // end catch
36             catch ( System.IO.IOException ioError )
37             {
38                 // compilation error--exception caught by preceding catch
39                 Console.WriteLine( ioError.Message );
40                 Console.WriteLine(
41                     "Perform logic for handling IOException\n" );
42
43                 // perform logic for handling IOException
44             } // end catch

```

```

45     catch ( FormatException formatError )
46     {
47         // compilation error--exception caught by preceding catch
48         Console.WriteLine( formatError.Message );
49         Console.WriteLine( "Perform logic for handling " +
50             "FormatException\n" );
51
52         // perform logic for handling FormatException
53     } // end catch
54     catch ( ArithmeticException arithmeticError )
55     {
56         // compilation error--exception caught by preceding catch
57         Console.WriteLine( arithmeticError.Message );
58         Console.WriteLine( "Perform logic for handling " +
59             "ArithmeticException\n" );
60
61         // perform logic for handling ArithmeticException
62     } // end catch
63 } // end for
64 } // end Main
65 } // end class OrderTest

```

Error List					
3 Errors 0 Warnings 0 Messages					
	Description	File	Line	Column	Project
1	A previous catch clause already catches all exceptions of this or of a super type ('System.Exception')	OrderTest.cs	36	18	OrderTest
2	A previous catch clause already catches all exceptions of this or of a super type ('System.Exception')	OrderTest.cs	45	18	OrderTest
3	A previous catch clause already catches all exceptions of this or of a super type ('System.Exception')	OrderTest.cs	54	18	OrderTest

13.6 Exceptions can be used to indicate problems that occur when an object is being constructed. Write a program that shows a constructor passing information about constructor failure to an exception handler. The exception thrown also should contain the arguments sent to the constructor.

ANS:

```

1  // Exercise 13.6 Solution: ConstructorException.cs
2  // Passes information to exception handler from constructor.
3  using System;
4
5  // class for throwing exceptions
6  class ThrowException
7  {
8      // throws an exception using arguments
9      public ThrowException( int i, int j )
10     {
11         throw new Exception( "Exception thrown from " +
12             "constructor with arguments " + i + " " + j );
13     } // end constructor
14 } // end class ThrowException

```

```

15
16 // use class ThrowException to throw exception
17 public class ConstructorException
18 {
19     public static void Main( string[] args )
20     {
21         ThrowException throwableException;
22
23         // throw and catch custom exception
24         try
25         {
26             throwableException = new ThrowException( 5, 3 );
27         } // end try
28         catch ( Exception e ) // catch exception thrown in try block
29         {
30             Console.WriteLine( e.Message );
31         } // end catch
32     } // end Main
33 } // end class ConstructorException

```

Exception thrown from constructor with arguments 5 3

13.7 Write a program that demonstrates rethrowing an exception.

ANS:

```

1 // Exercise 13.7 Solution: Rethrow.cs
2 // Demonstrating how to rethrow an exception.
3 using System;
4
5 class Rethrow
6 {
7     public static void Main( string[] args )
8     {
9         // call method SomeMethod, catch any thrown exceptions
10        try
11        {
12            SomeMethod();
13        } // end try
14        catch ( Exception e )
15        {
16            Console.WriteLine( "Handled in Main: " + e );
17        } // end catch
18    } // end Main
19
20    // call SomeMethod2
21    private static void SomeMethod()
22    {
23        // call SomeMethod2, catch and rethrow any thrown exceptions
24        try
25        {

```

```

26         SomeMethod2();
27     } // end try
28     catch ( Exception )
29     {
30         throw; // rethrow exception
31     } // end catch
32 } // end method SomeMethod
33
34 // throw general exception
35 private static void SomeMethod2()
36 {
37     throw new Exception();
38 } // end method SomeMethod2
39 } // end class Rethrow

```

Handled in Main: System.Exception: Exception of type 'System.Exception' was thrown.

at Rethrow.SomeMethod2() in C:\solutions\sol_Ch13\Ex13_07\Rethrow\Rethrow.cs:line 37

at Rethrow.SomeMethod() in C:\solutions\sol_Ch13\Ex13_07\Rethrow\Rethrow.cs:line 30

at Rethrow.Main(String[] args) in C:\solutions\sol_Ch13\Ex13_07\Rethrow\Rethrow.cs:line 12

13.8 Write a program demonstrating that a method with its own try block does not have to catch every possible exception that occurs within the try block—some exceptions can slip through to, and be handled in, other scopes.

ANS: [Note: Since this is a mechanical exercise, we did not provide the complete set of constructors for the exception class.]

```

1 // Exercise 13.8 Solution: OtherScopes.cs
2 // Allowing other scopes to handle exceptions.
3 using System;
4
5 // Exception derived class
6 class ExceptionA : Exception
7 {
8     // empty body
9 } // end class ExceptionA
10
11 // allow other scopes to handle exceptions
12 class OtherScopes
13 {
14     public static void Main( string[] args )
15     {
16         // call method SomeMethod, catch any thrown exceptions
17         try
18         {
19             SomeMethod();
20         } // end try

```

```

21     catch ( Exception e )
22     {
23         Console.WriteLine( "Handled in Main: " + e );
24     } // end catch
25 } // end Main
26
27 // call SomeMethod2
28 private static void SomeMethod()
29 {
30     // call SomeMethod2, catch any thrown exceptions
31     try
32     {
33         SomeMethod2();
34     }
35     catch ( ExceptionA e ) // catch exception of type ExceptionA
36     {
37         // does not catch Exception; just catches ExceptionA
38         Console.WriteLine( "Handled ExceptionA in SomeMethod: " + e );
39     } // end try
40 } // end method SomeMethod
41
42 // throw general exception
43 private static void SomeMethod2()
44 {
45     throw new Exception();
46 } // end method SomeMethod2
47 } // end class OtherScopes

```

```

Handled in Main: System.Exception: Exception of type 'System.Exception' was
thrown.
   at OtherScopes.SomeMethod2() in C:\solutions\sol_Ch13\Ex13_08\
OtherScopes\OtherScopes.cs:line 45
   at OtherScopes.SomeMethod() in C:\solutions\sol_Ch13\Ex13_08\
OtherScopes\OtherScopes.cs:line 33
   at OtherScopes.Main(String[] args) in C:\solutions\sol_Ch13\Ex13_08\Other-
Scopes\OtherScopes.cs:line 19

```

13.9 Write a program that throws an exception from a deeply nested method. The catch block should follow the try block that encloses the call chain. The exception caught should be one you defined yourself. In catching the exception, display the exception's message and stack trace.

ANS:

```

1 // Exercise 13.9 Solution: TestException.cs
2 // Defining an Exception.
3 using System;
4
5 public class TestException : Exception
6 {

```

```

7    // default constructor
8    public TestException()
9        : base( "This is a tester exception." )
10    {
11        // empty body
12    } // end default constructor
13
14    // constructor for customizing error message
15    public TestException( string messageValue )
16        : base( messageValue )
17    {
18        // empty body
19    } // end one-argument constructor
20
21    // constructor for customizing error message and specifying
22    // InnerException object
23    public TestException( string messageValue, Exception inner )
24        : base( messageValue, inner )
25    {
26        // empty body
27    } // end two-argument constructor
28 } // end class TestException

```

```

1  // Exercise 13.9 Solution: TestExceptionTest.cs
2  // Testing the defined exception from a deeply nested call.
3  using System;
4
5  class TestExceptionTest
6  {
7      static void F()
8      {
9          throw new TestException();
10     } // end method F
11
12     static void G()
13     {
14         F();
15     } // end method G
16
17     static void H()
18     {
19         G();
20     } // end method H
21
22     static void I()
23     {
24         H();
25     } // end method I
26
27     static void J()
28     {
29         I();
30     } // end method J

```

```

31
32     static void K()
33     {
34         J();
35     } // end method K
36
37     public static void Main( string[] args )
38     {
39         try
40         {
41             K();
42         } // end try
43         catch ( TestException test )
44         {
45             Console.WriteLine( test.Message + "\n" );
46             Console.WriteLine( "The stack trace for this exception is:" );
47             Console.WriteLine( test.StackTrace );
48         } // end catch
49     } // end Main
50 } // end class TestExceptionTest

```

This is a tester exception.

The stack trace for this exception is:
 at TestExceptionTest.F() in C:\solutions\sol_Ch13\Ex13_09\TestException\TestExceptionTest.cs:line 9
 at TestExceptionTest.G() in C:\solutions\sol_Ch13\Ex13_09\TestException\TestExceptionTest.cs:line 14
 at TestExceptionTest.H() in C:\solutions\sol_Ch13\Ex13_09\TestException\TestExceptionTest.cs:line 19
 at TestExceptionTest.I() in C:\solutions\sol_Ch13\Ex13_09\TestException\TestExceptionTest.cs:line 24
 at TestExceptionTest.J() in C:\solutions\sol_Ch13\Ex13_09\TestException\TestExceptionTest.cs:line 29
 at TestExceptionTest.K() in C:\solutions\sol_Ch13\Ex13_09\TestException\TestExceptionTest.cs:line 34
 at TestExceptionTest.Main(String[] args) in C:\solutions\sol_Ch13\Ex13_09\TestException\TestExceptionTest.cs:line 41

13.10 Create an application that inputs miles driven and gallons used, and calculates miles per gallon. The example should use exception handling to process the `FormatExceptions` that occur when converting the input strings to doubles. If invalid data is entered, display a message informing the user.

ANS:

```

1  // Exercise 13.10 Solution: MilesPerGallon.cs
2  // Handling FormatExceptions for a miles per gallon example.
3  using System;
4
5  public class MilesPerGallon
6  {
7      public static void Main( string[] args )
8      {

```

```

9      string quit;
10
11      do
12      {
13          try
14          {
15              // retrieve user input and display miles per gallon
16              Console.Write( "Enter miles driven: " );
17              double miles = Convert.ToDouble( Console.ReadLine() );
18
19              Console.Write( "Enter gallons used: " );
20              double gallons = Convert.ToDouble( Console.ReadLine() );
21
22              Console.WriteLine( "{0:F} miles per gallon.",
23                  CalculateMPG( miles, gallons ) );
24          } // end try
25          catch ( FormatException )
26          {
27              Console.WriteLine( "Please enter decimal numbers for " +
28                  "the miles driven and gallons used." );
29          } // end catch
30
31          Console.Write( "Quit (yes/no)? " );
32          quit = Console.ReadLine().ToLower();
33          } while ( quit != "yes" );
34      } // end Main
35
36      // calculate and return miles per gallon
37      private static double CalculateMPG( double milesDriven,
38          double gallonsUsed )
39      {
40          return milesDriven / gallonsUsed;
41      } // end method MilesPerGallon
42  } // end class MilesPerGallon

```

```

Enter miles driven: 300.7
Enter gallons used: 15.2
19.78 miles per gallon.
Quit (yes/no)? no
Enter miles driven: 408.5
Enter gallons used: hello
Please enter decimal numbers for the miles driven and gallons used.
Quit (yes/no)? yes

```

14

Graphical User Interfaces with Windows Forms: Practical Solutions

*... the wisest prophets make sure
of the event first.*

—Horace Walpole

*... The user should feel in control
of the computer; not the other
way around. This is achieved in
applications that embody three
qualities: responsiveness,
permissiveness, and consistency.*

—*Inside Macintosh, Volume 1*

Apple Computer, Inc. 1985

*All the better to see you with my
dear.*

—The Big Bad Wolf to Little Red Riding
Hood

Objectives

In this chapter you'll learn:

- Design principles of graphical user interfaces (GUIs).
- How to create graphical user interfaces.
- How to process events in response to user interactions with GUI controls.
- The namespaces that contain the classes for GUI controls and event handling.
- How to create and manipulate various controls.
- How to add descriptive ToolTips to GUI controls.
- How to process mouse and keyboard events.



Self-Review Exercises

14.1 State whether each of the following is *true* or *false*. If *false*, explain why.

a) The `KeyData` property includes data about modifier keys.

ANS: True.

b) Windows Forms commonly are used to create GUIs.

ANS: True.

c) A `Form` is a container.

ANS: True.

d) All `Forms`, components and controls are classes.

ANS: True.

e) `CheckBoxes` are used to represent a set of mutually exclusive options.

ANS: False. `RadioButtons` are used to represent a set of mutually exclusive options.

f) A `Label` displays text that a user running an application can edit.

ANS: False. A `Label`'s text cannot be edited by the user.

g) Button presses generate events.

ANS: True.

h) All mouse events use the same event-arguments class.

ANS: False. Some mouse events use `EventArgs`, others use `MouseEventArgs`.

i) Visual Studio can register an event and create an empty event handler.

ANS: True.

j) The `NumericUpDown` control is used to specify a range of input values.

ANS: True.

k) A control's tool-tip text is set with the `ToolTip` property of class `Control`.

ANS: False. A control's tool-tip text is set using a `ToolTip` component that must be added to the application.

14.2 Fill in the blanks in each of the following statements:

a) The active control is said to have the _____.

ANS: focus.

b) The `Form` acts as a(n) _____ for the controls that are added.

ANS: container.

c) GUIs are _____ driven.

ANS: event.

d) Every method that handles the same event must have the same _____.

ANS: signature.

e) A(n) _____ `TextBox` masks user input with a character used repeatedly.

ANS: password.

f) Class _____ and class _____ help arrange controls on a GUI and provide logical groups for radio buttons.

ANS: `GroupBox`, `Panel`.

g) Typical mouse events include _____, _____ and _____.

ANS: mouse clicks, mouse presses, mouse moves.

h) _____ events are generated when a key on the keyboard is pressed or released.

ANS: Key.

i) The modifier keys are _____, _____ and _____.

ANS: *Shift*, *Ctrl*, *Alt*.

j) A(n) _____ event or delegate can call multiple methods.

ANS: multicast.

Exercises

14.3 Extend the program in Fig. 14.26 to include a `CheckBox` for every font-style option. [*Hint:* Use logical exclusive OR (^) rather than testing for every bit explicitly.]

ANS:

```

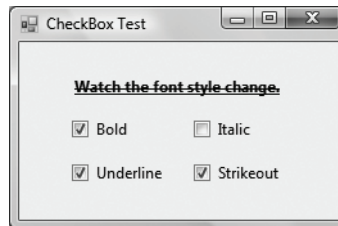
1  // Exercise 14.3: CheckBoxTestForm.cs
2  // Using CheckBoxes to toggle underline and strikethrough styles
3  // as well as italic and bold.
4  using System;
5  using System.Drawing;
6  using System.Windows.Forms;
7
8  namespace CheckBoxTest
9  {
10     public partial class CheckBoxTestForm : Form
11     {
12         // constructor
13         public CheckBoxTestForm()
14         {
15             InitializeComponent();
16         } // end constructor
17
18         // when the Bold CheckBox is clicked, make text bold if not bold;
19         // if already bold, make not bold
20         private void boldCheckBox_CheckedChanged(
21             object sender, EventArgs e )
22         {
23             outputLabel.Font =
24                 new Font( outputLabel.Font.Name, outputLabel.Font.Size,
25                     outputLabel.Font.Style ^ FontStyle.Bold );
26         } // end method boldCheckBox_CheckedChanged
27
28         // when the Italic CheckBox is clicked, make text
29         // italic if not italic; if already italic, make not italic
30         private void italicCheckBox_CheckedChanged(
31             object sender, EventArgs e )
32         {
33             outputLabel.Font =
34                 new Font( outputLabel.Font.Name, outputLabel.Font.Size,
35                     outputLabel.Font.Style ^ FontStyle.Italic );
36         } // end method italicCheckBox_CheckedChanged
37
38         // when the Underline CheckBox is clicked, make text underlined
39         // if not underlined; if already underlined, make not underlined
40         private void underlineCheckBox_CheckedChanged(
41             object sender, EventArgs e )
42         {
43             outputLabel.Font =
44                 new Font( outputLabel.Font.Name, outputLabel.Font.Size,
45                     outputLabel.Font.Style ^ FontStyle.Underline );
46         } // end method underlineCheckBox_CheckedChanged
47

```

```

48      // when the Strikeout CheckBox is clicked, strike out text
49      // if not striked out; if already striked out make, not striked out
50      private void strikeoutCheckBox_CheckedChanged(
51          object sender, EventArgs e )
52      {
53          outputLabel.Font =
54              new Font( outputLabel.Font.Name, outputLabel.Font.Size,
55                  outputLabel.Font.Style ^ FontStyle.Strikeout );
56      } // end method strikeoutCheckBox_CheckedChanged
57  } // end class CheckBocTestForm
58 } // end namespace CheckBoxTest

```



14.4 Create the GUI in Fig. 14.41 (you do not have to provide functionality).

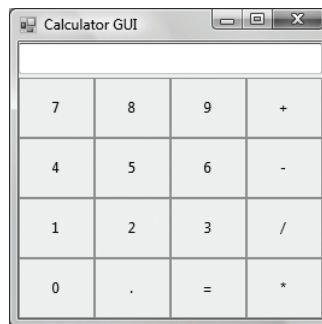


Fig. 14.41 | Calculator GUI.

ANS:

- Set the Form's Text to "Calculator GUI".
- Create a TextBox.
- Create a Button and modify its size so that it looks square.
- Copy and paste the Button 15 times to create the remaining buttons.
- Edit the text and location of all the Buttons so that your Form looks like the desired GUI.

14.5 Create the GUI in Fig. 14.42 (you do not have to provide functionality).

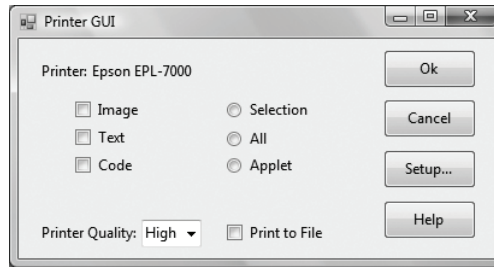


Fig. 14.42 | Printer GUI.

ANS:

- a) Set the Form's Text to "Printer GUI".
- b) Drag two Labels onto your Form. One should have the text "Printer" and the other the text "Epson EPL-7000". Drag one more label onto the bottom of the Form for "Print Quality".
- a) Create three CheckBoxes—for "Image", "Text" and "Code". Add one more checkbox on the bottom—for "Print to File".
- b) Create three RadioButtons—for "Selection", "All" and "Application".
- c) Create a ComboBox and change its text to read "High".
- d) Create four Buttons. Their text should be "Ok", "Cancel", "Setup..." and "Help".
- e) Resize and reposition all controls so that the Form looks like the desired GUI.

14.6 (*Temperature Conversions*) Write a temperature conversion program that converts from Fahrenheit to Celsius. The Fahrenheit temperature should be entered from the keyboard (via a TextBox). A Label should be used to display the converted temperature. Use the following formula for the conversion:

$$Celsius = (5 / 9) \times (Fahrenheit - 32)$$

ANS:

```

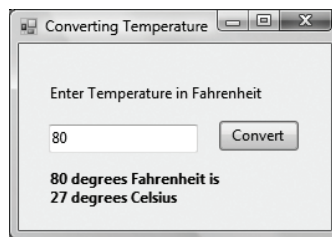
1 // Ex. 14.6: TemperatureForm.cs
2 // Converting temperatures from Fahrenheit to Celsius.
3 using System;
4 using System.Windows.Forms;
5
6 namespace Temperature
7 {
8     public partial class TemperatureForm : Form
9     {
10         // constructor
11         public TemperatureForm()
12         {
13             InitializeComponent();
14         } // end constructor
15     }

```

```

16 // event handler that converts temperature
17 private void convertButton_Click( object sender, EventArgs e )
18 {
19     int fahrenheit;
20     int celsius;
21
22     fahrenheit = Convert.ToInt32( fahrenheitTextBox.Text );
23     celsius = Convert.ToInt32( 5.0 / 9.0 * ( fahrenheit - 32 ) );
24
25     displayLabel.Text = fahrenheit +
26         " degrees Fahrenheit is\n" + celsius + " degrees Celsius";
27 } // end method convertButton_Click
28 } // end class TemperatureForm
29 } // end namespace Temperature

```



14.7 (Enhanced Painter) Extend the program of Fig. 14.38 to include options for changing the size and color of the lines drawn. Create a GUI similar to Fig. 14.43. The user should be able to draw on the application's Panel. To retrieve a Graphics object for drawing, call method *panel.Name.CreateGraphics()*, substituting in the name of your Panel.

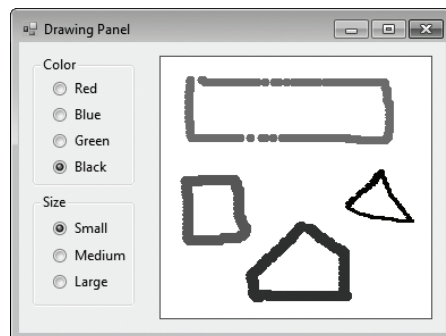


Fig. 14.43 | Drawing Panel GUI.

ANS:

```

1 // Ex. 14.7: DrawingForm.cs
2 // Drawing on the Form using multiple colors and sizes.
3 using System;

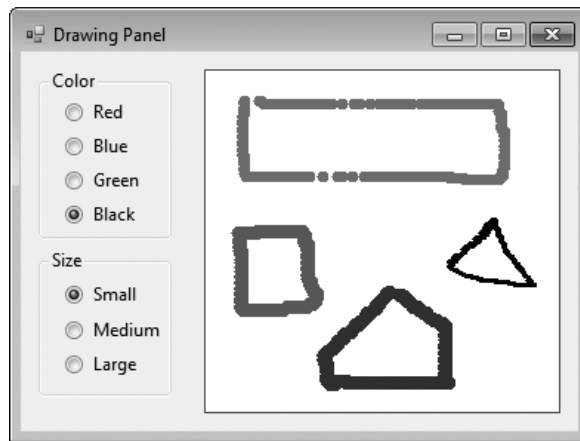
```

```
4 using System.Drawing;
5 using System.Windows.Forms;
6
7 namespace Drawing
8 {
9     public partial class DrawingForm : Form
10    {
11        bool shouldPaint = false; // determines whether to paint
12        int dotSize = 4; // default size (4)
13        Color brushColor; // color object
14
15        // constructor
16        public DrawingForm()
17        {
18            InitializeComponent();
19        } // end constructor
20
21        // changed brushColor to red
22        private void redRadioButton_CheckedChanged(
23            object sender, EventArgs e )
24        {
25            if ( redRadioButton.Checked )
26                brushColor = Color.Red;
27        } // end method redRadioButton_CheckedChanged
28
29        // changed brushColor to blue
30        private void blueRadioButton_CheckedChanged(
31            object sender, EventArgs e )
32        {
33            if ( blueRadioButton.Checked )
34                brushColor = Color.Blue;
35        } // end method blueRadioButton_CheckedChanged
36
37        // changed brushColor to green
38        private void greenRadioButton_CheckedChanged(
39            object sender, EventArgs e )
40        {
41            if ( greenRadioButton.Checked )
42                brushColor = Color.Green;
43        } // end method greenRadioButton_CheckedChanged
44
45        // changed brushColor to black
46        private void blackRadioButton_CheckedChanged(
47            object sender, EventArgs e )
48        {
49            if ( blackRadioButton.Checked )
50                brushColor = Color.Black;
51        } // end method blackRadioButton_CheckedChanged
52
53        // changes dotSize to small size
54        private void smallRadioButton_CheckedChanged(
55            object sender, EventArgs e )
56        {
```

```

57         if ( smallRadioButton.Checked )
58             dotSize = 4;
59     } // end method smallRadioButton_CheckedChanged
60
61     // changes dotSize to medium size
62     private void mediumRadioButton_CheckedChanged(
63         object sender, EventArgs e )
64     {
65         if ( mediumRadioButton.Checked )
66             dotSize = 8;
67     } // end method mediumRadioButton_CheckedChanged
68
69     // changes dotSize to large size
70     private void largeRadioButton_CheckedChanged(
71         object sender, EventArgs e )
72     {
73         if ( largeRadioButton.Checked )
74             dotSize = 10;
75     } // end method largeRadioButton_CheckedChanged
76
77     // should paint when mouse button is pressed down
78     private void drawingPanel_MouseDown(
79         object sender, MouseEventArgs e )
80     {
81         shouldPaint = true;
82     } // end method drawingPanel_MouseDown
83
84     // stop painting when mouse button is released
85     private void drawingPanel_MouseUp(
86         object sender, MouseEventArgs e )
87     {
88         shouldPaint = false;
89     } // end method drawingPanel_MouseUp
90
91     // draw circle whenever mouse button moves and is held down
92     private void drawingPanel_MouseMove(
93         object sender, MouseEventArgs e )
94     {
95         if ( shouldPaint ) // check if mouse button is being pressed
96         {
97             // draw a circle where the mouse pointer is present
98             using ( Graphics graphics = drawingPanel.CreateGraphics() )
99             {
100                 graphics.FillEllipse(
101                     new SolidBrush(brushColor), e.X, e.Y, dotSize, dotSize);
102             } // end using; calls graphics.Dispose()
103         } // end if
104     } // end method drawingPanel_MouseMove
105 } // end class DrawingForm
106 } // end namespace Drawing

```



14.8 (*Guess the Number Game*) Write a program that plays “guess the number” as follows: Your program chooses the number to be guessed by selecting an `int` at random in the range 1–1000. The program then displays the following text in a label:

I have a number between 1 and 1000--can you guess my number?
Please enter your first guess.

A `TextBox` should be used to input the guess. As each guess is input, the background color should change to red or blue. Red indicates that the user is getting “warmer,” blue that the user is getting “colder.” A `Label` should display either “Too High” or “Too Low,” to help the user zero in on the correct answer. When the user guesses the correct answer, display “Correct!” in a message box, change the Form’s background color to green and disable the `TextBox`. Recall that a `TextBox` (like other controls) can be disabled by setting the control’s `Enabled` property to `false`. Provide a `Button` that allows the user to play the game again. When the `Button` is clicked, generate a new random number, change the background to the default color and enable the `TextBox`.

ANS:

```

1  // Ex. 14.8: GuessForm.cs
2  // Demonstrating a guessing game that uses GUI.
3  using System;
4  using System.Drawing;
5  using System.Windows.Forms;
6
7  namespace Guess
8  {
9      public partial class GuessForm : Form
10     {
11         int secret = 0;
12         int lastGuess = 0;
13         bool firstGuess = true;
14
15         // constructor
16         public GuessForm()
17         {

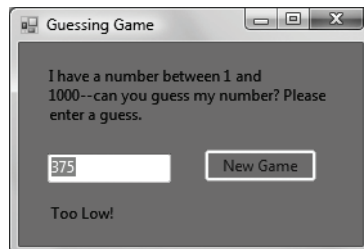
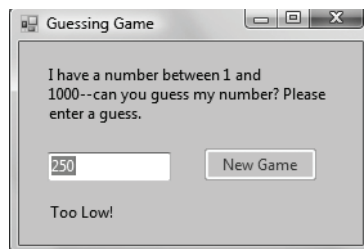
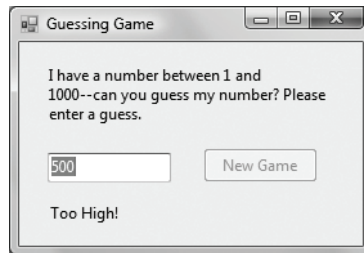
```

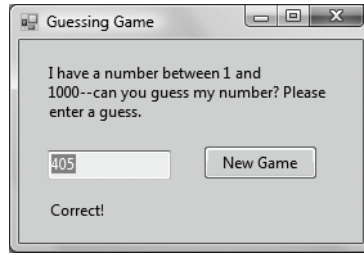
```
18     InitializeComponent();
19 } // end constructor
20
21 // randomly generate secret number (1-1000)
22 private void GenerateSecret()
23 {
24     Random randomNumber = new Random();
25     secret = randomNumber.Next( 1000 ) + 1;
26 } // end method GenerateSecret
27
28 private int CheckGuess( int user )
29 {
30     if ( user > secret ) // too high
31         return 1;
32     if ( user < secret ) // too low
33         return -1;
34     else // correct
35         return 0;
36 } // end method CheckGuess
37
38 private void inputTextBox_KeyDown( object sender, KeyEventArgs e )
39 {
40     // allow user to press Enter in textbox
41     if ( e.KeyCode == Keys.Enter )
42     {
43         int userGuess = Convert.ToInt32( inputTextBox.Text );
44
45         // check if user is warmer or colder
46         if ( !firstGuess )
47         {
48             if ( Math.Abs( userGuess - secret ) <
49                 Math.Abs( lastGuess - secret ) )
50                 BackColor = Color.Red;
51             else
52                 BackColor = Color.LightBlue;
53         }
54
55         firstGuess = false;
56
57         // CheckGuess returns 0 if correct, -1 if too low,
58         // 1 if too high
59         int rightOrWrong = CheckGuess( userGuess );
60
61         // if guess is right, allow user to play again
62         if ( rightOrWrong == 0 )
63         {
64             outputLabel.Text = "Correct!";
65             newGameButton.Enabled = true;
66             inputTextBox.ReadOnly = true;
67             BackColor = Color.LightGreen;
68             firstGuess = true;
69         } // end if
70         else if ( rightOrWrong == -1 )
71             outputLabel.Text = "Too Low!";
```

```

72         else
73             outputLabel.Text = "Too High!";
74
75             // clear guess
76             inputTextBox.SelectAll();
77             lastGuess = userGuess;
78         } // end if
79     } // end method inputTextBox_KeyDown
80
81     private void newGameButton_Click( object sender, EventArgs e )
82     {
83         outputLabel.Text = "New Game Started!";
84
85         GenerateSecret();
86
87         newGameButton.Enabled = false;
88         inputTextBox.ReadOnly = false;
89         BackColor = Color.FromName( "Control" );
90     } // end method newGameButton_Click
91 } // end class GuessForm
92 } // end namespace Guess

```





14.9 (Fuzzy Dice Order Form) Write an application that allows users to process orders for fuzzy dice. The application should calculate the total price of the order, including tax and shipping. TextBoxes for inputting the order number, the customer name and the shipping address are provided. Initially, these fields contain text that describes their purpose. Provide CheckBoxes for selecting the fuzzy-dice color and TextBoxes for inputting the quantities of fuzzy dice to order. The application should update the total cost, tax and shipping when the user changes any one of the three **Quantity** fields' values. The application should also contain a Button that when clicked, returns all fields to their original values. Use 5% for the tax rate. Shipping charges are \$1.50 for up to 20 pairs of dice. If more than 20 pairs of dice are ordered, shipping is free. All fields must be filled out at the top, and an item must be checked for the user to enter a quantity for that item.

ANS:

```

1 // Exercise 14.9 Solution: FuzzyDiceForm.cs
2 // Application that allows user to order fuzzy dice,
3 // specifying the type and amount of dice.
4 using System;
5 using System.Windows.Forms;
6
7 namespace FuzzyDice
8 {
9     public partial class FuzzyDiceForm : Form
10    {
11        // constructor
12        public FuzzyDiceForm()
13        {
14            InitializeComponent();
15        } // end constructor
16
17        private void whiteBlackTextBox_TextChanged(
18            object sender, EventArgs e )
19        {
20            // store quantity entered as int
21            int numberOfWhiteBlack = Convert.ToInt32(
22                whiteBlackTextBox.Text );
23
24            // display message if user tries to enter a value
25            // without selecting CheckBox
26
27            if ( numberOfWhiteBlack != 0 &&
28                whiteBlackCheckBox.Checked == false )
29            {

```

```

30         // keep white/black quantity at 0
31         whiteBlackTextBox.Text = "0";
32
33         MessageBox.Show(
34             "Please check item you wish to purchase",
35             "No Item Selected", MessageBoxButtons.OK,
36             MessageBoxIcon.Exclamation );
37     } // end if
38     // display message if shipping information is not supplied
39     else if ( string.IsNullOrEmpty( orderNumberTextBox.Text ) ||
40         string.IsNullOrEmpty( nameTextBox.Text ) ||
41         string.IsNullOrEmpty( addressLine1TextBox.Text ) ||
42         string.IsNullOrEmpty( addressLine2TextBox.Text ) )
43     {
44         // display message in dialog
45         MessageBox.Show(
46             "Please fill out all information fields.",
47             "Empty Fields", MessageBoxButtons.OK,
48             MessageBoxIcon.Exclamation );
49     } // end else if
50     // display message if negative number entered
51     else if ( numberOfWhiteBlack < 0 )
52     {
53         whiteBlackTextBox.Text = "0";
54         MessageBox.Show(
55             "Please enter a positive quantity",
56             "Bad Input", MessageBoxButtons.OK,
57             MessageBoxIcon.Exclamation );
58     } // end else if
59     else // calculate totals
60     {
61         CalculateAndDisplayTotals();
62     } // end else
63 } // end method whiteBlackTextBox_TextChanged
64
65 private void redBlackTextBox_TextChanged(
66     object sender, EventArgs e )
67 {
68     // store quantity entered as int
69     int numberOfRedBlack = Convert.ToInt32( redBlackTextBox.Text );
70
71     // display message if user tries to enter a value
72     // without selecting CheckBox
73
74     if ( numberOfRedBlack != 0 &&
75         redBlackCheckBox.Checked == false )
76     {
77         // keep white/black quantity at 0
78         redBlackTextBox.Text = "0";
79
80         MessageBox.Show(
81             "Please check item you wish to purchase",
82             "No Item Selected", MessageBoxButtons.OK,
83             MessageBoxIcon.Exclamation );
84     } // end if

```

```

85         // display message if shipping information is not supplied
86     else if ( string.IsNullOrEmpty( orderNumberTextBox.Text ) ||
87         string.IsNullOrEmpty( nameTextBox.Text ) ||
88         string.IsNullOrEmpty( addressLine1TextBox.Text ) ||
89         string.IsNullOrEmpty( addressLine2TextBox.Text ) )
90     {
91         // display message in dialog
92         MessageBox.Show(
93             "Please fill out all information fields.",
94             "Empty Fields", MessageBoxButtons.OK,
95             MessageBoxIcon.Exclamation );
96     } // end else if
97     // display message if negative number entered
98     else if ( numberOfRedBlack < 0 )
99     {
100         redBlackTextBox.Text = "0";
101         MessageBox.Show(
102             "Please enter a positive quantity",
103             "Bad Input", MessageBoxButtons.OK,
104             MessageBoxIcon.Exclamation );
105     } // end else if
106     else // calculate totals
107     {
108         CalculateAndDisplayTotals();
109     } // end else
110 } // end method redBlackTextBox_TextChanged
111
112 private void blueBlackTextBox_TextChanged(
113     object sender, EventArgs e )
114 {
115     // store quantity entered as int
116     int numberOfBlueBlack = Convert.ToInt32(
117         blueBlackTextBox.Text );
118
119     // display message if user tries to enter a value
120     // without selecting CheckBox
121     if ( numberOfBlueBlack != 0 &&
122         blueBlackCheckBox.Checked == false )
123     {
124         // keep white/black quantity at 0
125         blueBlackTextBox.Text = "0";
126
127         MessageBox.Show(
128             "Please check item you wish to purchase",
129             "No Item Selected", MessageBoxButtons.OK,
130             MessageBoxIcon.Exclamation );
131     } // end if
132     // display message if shipping information is not supplied
133     else if ( string.IsNullOrEmpty( orderNumberTextBox.Text ) ||
134         string.IsNullOrEmpty( nameTextBox.Text ) ||
135         string.IsNullOrEmpty( addressLine1TextBox.Text ) ||
136         string.IsNullOrEmpty( addressLine2TextBox.Text ) )
137     {

```

```

138         // display message in dialog
139         MessageBox.Show(
140             "Please fill out all information fields.",
141             "Empty Fields", MessageBoxButtons.OK,
142             MessageBoxIcon.Exclamation );
143     } // end else if
144     // display message if negative number entered
145     else if ( numberOfBlueBlack < 0 )
146     {
147         blueBlackTextBox.Text = "0";
148         MessageBox.Show(
149             "Please enter a positive quantity",
150             "Bad Input", MessageBoxButtons.OK,
151             MessageBoxIcon.Exclamation );
152     } // end else if
153     else // calculate totals
154     {
155         CalculateAndDisplayTotals();
156     } // end else
157 } // end method blueBlackTextBox_TextChanged
158
159 // clear all fields
160 private void clearButton_Click( object sender, EventArgs e )
161 {
162     // set all fields to their original values
163     orderNumberTextBox.Text = "0";
164     nameTextBox.Text = "Enter name here";
165     addressLine1TextBox.Text = "Address Line 1";
166     addressLine2TextBox.Text = "Address Line 2";
167     cityStateZipTextBox.Text = "City, State, zip";
168     whiteBlackTextBox.Text = "0";
169     redBlackTextBox.Text = "0";
170     blueBlackTextBox.Text = "0";
171     whiteBlackLabel1.Text = "$0.00";
172     redBlackLabel1.Text = "$0.00";
173     blueBlackLabel1.Text = "$0.00";
174     subtotalLabel1.Text = "$0.00";
175     taxLabel1.Text = "$0.00";
176     shippingLabel1.Text = "$0.00";
177     totalLabel1.Text = "$0.00";
178     whiteBlackCheckBox.Checked = false;
179     redBlackCheckBox.Checked = false;
180     blueBlackCheckBox.Checked = false;
181 } // end method clearButton_Click
182
183 private void whiteBlackCheckBox_CheckedChanged(
184     object sender, EventArgs e )
185 {
186     whiteBlackTextBox.Text = "0";
187     whiteBlackLabel1.Text = "0";
188     CalculateAndDisplayTotals();
189 } // end method whiteBlackCheckBox_CheckedChanged
190

```

```

191 private void redBlackCheckBox_CheckedChanged(
192     object sender, EventArgs e )
193 {
194     redBlackTextBox.Text = "0";
195     redBlackLabel.Text = "0";
196     CalculateAndDisplayTotals();
197 } // end method redBlackCheckBox_CheckedChanged
198
199 private void blueBlackCheckBox_CheckedChanged(
200     object sender, EventArgs e )
201 {
202     blueBlackTextBox.Text = "0";
203     blueBlackLabel.Text = "0";
204     CalculateAndDisplayTotals();
205 } // end method blueBlackCheckBox_CheckedChanged
206
207 private void CalculateAndDisplayTotals()
208 {
209     // individual totals
210     // total of white/black dice
211     decimal whiteBlackTotals =
212         Convert.ToDecimal( whiteBlackTextBox.Text ) * 6.25M;
213
214     // total of red/black dice
215     decimal redBlackTotals =
216         Convert.ToDecimal( redBlackTextBox.Text ) * 5M;
217
218     // total of blue/black dice
219     decimal blueBlackTotals =
220         Convert.ToDecimal( blueBlackTextBox.Text ) * 7.5M;
221
222     // display individual totals
223     whiteBlackLabel.Text = string.Format(
224         "{0:C}", whiteBlackTotals );
225     redBlackLabel.Text = string.Format( "{0:C}", redBlackTotals );
226     blueBlackLabel.Text = string.Format( "{0:C}", blueBlackTotals );
227
228     // subtotal, before tax and shipping
229     decimal subtotal =
230         whiteBlackTotals + redBlackTotals + blueBlackTotals;
231     subtotalLabel.Text = string.Format( "{0:C}", subtotal );
232
233     // calculate and display tax
234     decimal tax = subtotal * 0.05M;
235     taxLabel.Text = string.Format( "{0:C}", tax );
236
237     // shipping: $1.50 for up to 20 items
238     // free after 20 items
239     int numberOfItems = Convert.ToInt32( whiteBlackTextBox.Text ) +
240         Convert.ToInt32( redBlackTextBox.Text ) +
241         Convert.ToInt32( blueBlackTextBox.Text );
242     decimal shippingCost = 0M;
243
244     if ( numberOfItems <= 20 && numberOfItems > 0 )
245         shippingCost = 1.5M;

```

```

246
247         // display shipping cost
248         shippingLabel.Text = string.Format( "{0:C}", shippingCost );
249
250         // calculate and display total charge
251         decimal totalCharge = subtotal + tax + shippingCost;
252
253         totalLabel.Text = string.Format( "{0:C}", totalCharge );
254     } // end method CalculateAndDisplayTotals
255 } // end class FuzzyDiceForm
256 } // end namespace FuzzyDice

```

Type:	Quantity:	Price:	Totals:
<input checked="" type="checkbox"/> White/Black	2	\$6.25	\$12.50
<input type="checkbox"/> Red/Black	0	\$5.00	\$0.00
<input checked="" type="checkbox"/> Blue/Black	3	\$7.50	\$22.50
Subtotal:			\$35.00
Tax:			\$1.75
Shipping:			\$1.50
Total:			\$38.25

Making a Difference Exercises

14.10 (Ecofont) Ecofont (www.ecofont.eu/ecofont_en.html)—developed by SPRANQ (a Netherlands-based company)—is a free, open-source computer font designed to reduce by as much as 20% the amount of ink used for printing, thus reducing also the number of ink cartridges used and the environmental impact of the manufacturing and shipping processes (using less energy, less fuel for shipping, and so on). The font, based on sans-serif Verdana, has small circular “holes” in the letters that are not visible in smaller sizes—such as the 9- or 10-point type frequently used. Download Ecofont, then install the font file `Spranq_eco_sans_regular.ttf` using the instructions from the Ecofont website. Next, develop a GUI-based program that allows you to type text in a `TextBox` to be displayed in the Ecofont. Create **Increase Font Size** and **Decrease Font Size** buttons that allow you to scale up or down by one point at a time. Set the `TextBox`’s `Font` property to 9 point Ecofont. Set the `TextBox`’s `MultiLine` property to `true` so the user can enter multiple lines of text. As you scale up the font, you’ll be able to see the holes in the letters more clearly. As you scale down, the holes will be less apparent. To change the `TextBox`’s font programmatically, use a statement of the form:

```

inputTextBox.Font = new Font( inputTextBox.Font.FontFamily,
    inputTextBox.Font.SizeInPoints + 1 );

```

This changes the `TextBox`'s `Font` property to a new `Font` object that uses the `TextBox`'s current font, but adds 1 to its `SizeInPoints` property to increase the font size. A similar statement can be used to decrease the font size. What is the smallest font size at which you begin to notice the holes?

ANS:

```

1  // Exercise 14.10: EcofontForm.cs
2  // Demonstrating Ecofont
3  using System;
4  using System.Windows.Forms;
5  using System.Drawing;
6
7  namespace Ecofont
8  {
9      public partial class EcofontForm : Form
10     {
11         public EcofontForm()
12         {
13             InitializeComponent();
14         }
15
16         // add 1 to font point size in TextBox
17         private void increaseSizeButton_Click( object sender, EventArgs e )
18         {
19             inputTextBox.Font = new Font( inputTextBox.Font.FontFamily,
20                 inputTextBox.Font.SizeInPoints + 1 );
21         } // end method increaseSizeButton_Click
22
23         // subtract 1 to font point size in TextBox
24         private void decreaseSizeButton_Click( object sender, EventArgs e )
25         {
26             if ( inputTextBox.Font.SizeInPoints > 2 )
27                 inputTextBox.Font = new Font( inputTextBox.Font.FontFamily,
28                     inputTextBox.Font.SizeInPoints - 1 );
29         } // end method decreaseSizeButton_Click
30     } // end class EcofontForm
31 } // end namespace Ecofont

```



14.11 (*Project: Typing Tutor—Tuning a Crucial Skill in the Computer Age*) Typing quickly and correctly is an essential skill for working effectively with computers and the Internet. In this exercise, you'll build an application that can help users learn to "touch type" (i.e., type correctly without looking at the keyboard). The application should display a *virtual keyboard* that mimics the one on your computer and should allow the user to watch what he or she is typing on the screen without looking at the *actual keyboard*. Use Buttons to represent the keys. As the user presses each key, the application highlights the corresponding Button and adds the character to a TextBox that shows what the user has typed so far. [*Hint: To highlight a Button, use its BackColor property to change its background color. When the key is released, reset its original background color.*]

You can test your program by typing a pangram—a phrase that contains every letter of the alphabet at least once—such as "The quick brown fox jumped over a lazy dog." You can find other pangrams on the web.

To make the program more interesting you could monitor the user's accuracy. You could have the user type specific phrases that you've prestored in your program and that you display on the screen above the virtual keyboard. You could keep track of how many keystrokes the user types correctly and how many are typed incorrectly. You could also keep track of which keys the user is having difficulty with and display a report showing those keys.

ANS: No answer provided for this project.



Graphical User Interfaces with Windows Forms: Part 2 Solutions

15

*I claim not to have controlled
events, but confess plainly that
events have controlled me.*

—Abraham Lincoln

Capture its reality in paint!

—Paul Cézanne

*An actor entering through the
door, you've got nothing. But if
he enters through the window,
you've got a situation.*

—Billy Wilder

*But, soft! what light through
yonder window breaks?
It is the east, and Juliet is
the sun!*

—William Shakespeare

Objectives

In this chapter you'll learn:

- To create menus, tabbed windows and multiple document interface (MDI) programs.
- To use the `Listview` and `TreeView` controls for displaying information.
- To create hyperlinks using the `LinkLabel` control.
- To display lists of information in `ListBox` and `ComboBox` controls.
- To input date and time data with the `DateTimePicker`.
- To create custom controls.

Self-Review Exercises

- 15.1** State whether each of the following is *true* or *false*. If *false*, explain why.
- a) Menus provide groups of related classes.
ANS: False. Menus provide groups of related commands.
 - b) Menu items can display ComboBoxes, checkmarks and access shortcuts.
ANS: True.
 - c) The ListBox control allows only single selection (like a RadioButton).
ANS: False. Both controls can have single or multiple selection.
 - d) A ComboBox control typically has a drop-down list.
ANS: True.
 - e) Deleting a parent node in a TreeView control deletes its child nodes.
ANS: True.
 - f) The user can select only one item in a ListView control.
ANS: False. The user can select one or more items.
 - g) A TabPage can act as a container for RadioButtons.
ANS: True.
 - h) An MDI child window can have MDI children.
ANS: False. Only an MDI parent window can have MDI children. An MDI parent window cannot be an MDI child.
 - i) MDI child windows can be moved outside the boundaries of their parent window.
ANS: False. MDI child windows cannot be moved outside their parent window.
 - j) There are two basic ways to create a customized control.
ANS: False. There are three ways: 1) Derive from an existing control, 2) use a UserControl or 3) derive from Control and create a control from scratch.
- 15.2** Fill in the blanks in each of the following statements:
- a) Method _____ of class Process can open files and Web pages, similar to the Run... command in Windows.
ANS: Start.
 - b) If more elements appear in a ComboBox than can fit, a(n) _____ appears.
ANS: scrollbar.
 - c) The top-level node in a TreeView is the _____ node.
ANS: root.
 - d) A(n) _____ and a(n) _____ can display icons contained in an ImageList control.
ANS: ListView, TreeView.
 - e) The _____ property allows a menu to display a list of active child windows.
ANS: MdiList.
 - f) Class _____ allows you to combine several controls into a single, custom control.
ANS: UserControl.
 - g) The _____ saves space by layering TabPages on top of each other.
ANS: TabControl.
 - h) The _____ window layout option makes all MDI windows the same size and layers them so every title bar is visible (if possible).
ANS: Cascade.
 - i) _____ are typically used to display hyperlinks to other resources, files or Web pages.
ANS: LinkLabels.

Exercises

NOTE: Solutions to the programming exercises are located in the sol_ch15 folder.

16

Strings, Characters and Regular Expressions: Solutions

*The chief defect of Henry King
Was chewing little bits of string.*

—Hilaire Belloc

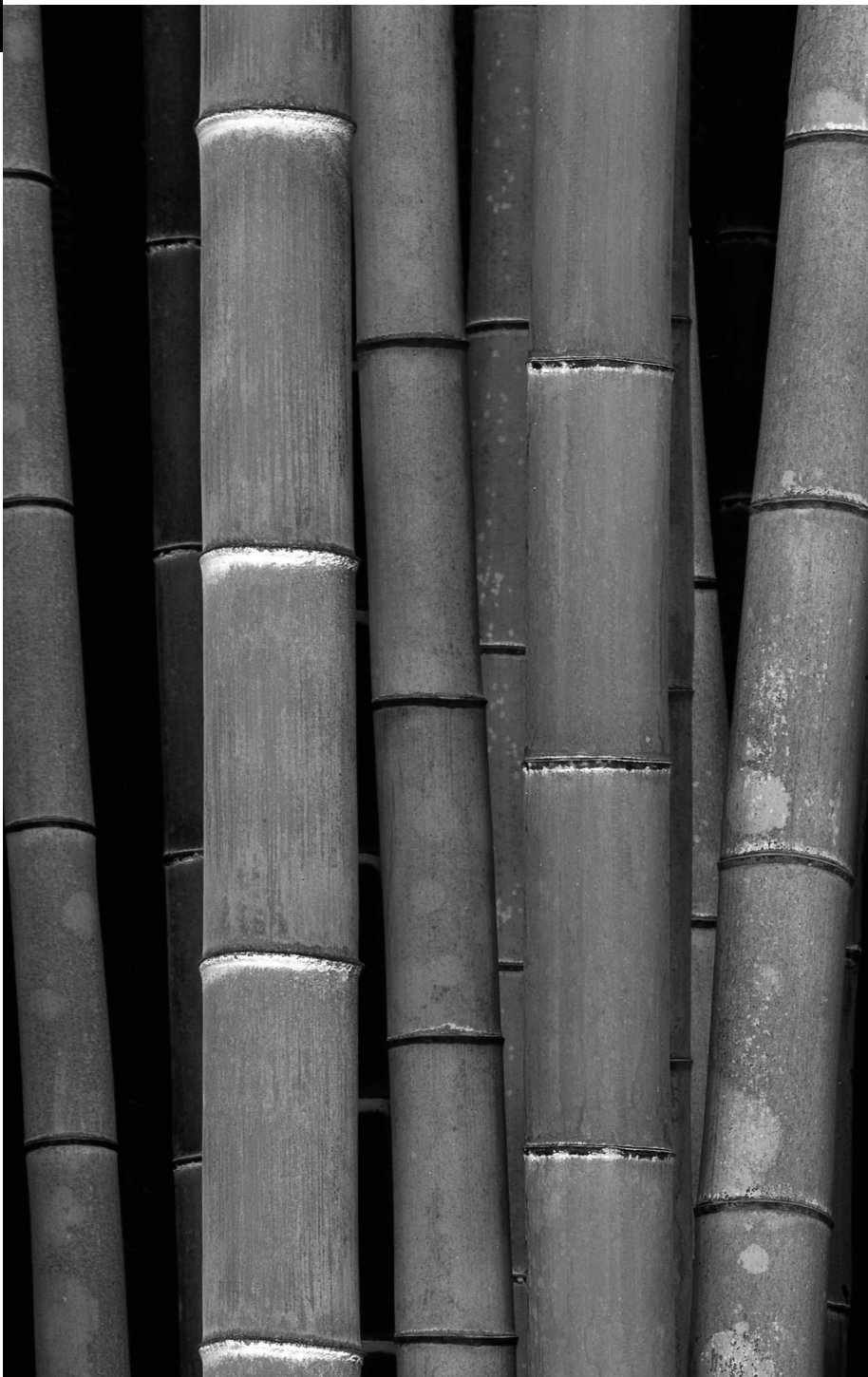
*The difference between the
almost-right word and the right
word is really a large matter—
it's the difference between the
lightning bug and the lightning.*

—Mark Twain

Objectives

In this chapter you'll learn:

- To create and manipulate immutable character-string objects of class `string` and mutable character-string objects of class `StringBuilder`.
- To manipulate character objects of struct `Char`.
- To use regular-expression classes `Regex` and `Match`.
- To iterate through matches to a regular expression.
- To use character classes to match any character from a set of characters.
- To use quantifiers to match a pattern multiple times.
- To search for patterns in text using regular expressions.
- To validate data using regular expressions and LINQ.
- To modify `strings` using regular expressions and class `Regex`.



Self-Review Exercises and Answers

- 16.1** State whether each of the following is *true* or *false*. If *false*, explain why.
- a) When strings are compared with `==`, the result is true if the strings contain the same values.
ANS: True.
 - b) A string can be modified after it is created.
ANS: False. strings are immutable; they cannot be modified after they are created. `StringBuilder` objects can be modified after they are created.
 - c) `StringBuilder` method `EnsureCapacity` sets the `StringBuilder` instance's capacity to the argument's value.
ANS: False. `EnsureCapacity` sets the instance's capacity to double either the current capacity or the value of its argument, whichever is larger.
 - d) Method `Equals` and the equality operator work the same for strings.
ANS: True.
 - e) Method `Trim` removes all whitespace at the beginning and the end of a string.
ANS: True.
 - f) A regular expression matches a string to a pattern.
ANS: True.
 - g) It is always better to use strings, rather than `StringBuilders`, because strings containing the same value will reference the same object.
ANS: False. `StringBuilder` should be used if the string is to be modified.
 - h) string method `ToUpper` creates a new string with the first letter capitalized.
ANS: False. string method `ToUpper` creates a new string with all of its letters capitalized.
 - i) The expression `\d` in a regular expression denotes all letters.
ANS: False. The expression `\d` in a regular expression denotes all digits.
- 16.2** Fill in the blanks in each of the following statements:
- a) To concatenate strings, use the _____ operator, `StringBuilder` method _____ or string method _____.
ANS: `+`, `Append`, `Concat`.
 - b) Method `Compare` of class `string` uses a _____ comparison of strings.
ANS: lexicographical.
 - c) Class `Regex` is located in namespace _____.
ANS: `System.Text.RegularExpressions`.
 - d) `StringBuilder` method _____ first formats the specified string, then concatenates it to the end of the `StringBuilder`.
ANS: `AppendFormat`
 - e) If the arguments to a `Substring` method call are out of range, an _____ exception is thrown.
ANS: `ArgumentOutOfRangeException`.
 - f) `Regex` method _____ changes all occurrences of a pattern in a string to a specified string.
ANS: `Replace`.
 - g) A C in a format string means to output the number as _____.
ANS: currency.
 - h) Regular expression quantifier _____ matches zero or more occurrences of an expression.
ANS: `*`.
 - i) Regular expression operator _____ inside square brackets will not match any of the characters in that set of brackets.
ANS: `^`.

16.3 Write statements to accomplish each of the following tasks:

a) Create a regular expression to match either a five-letter word or five-digit number.

ANS: `Regex regex = new Regex(@"\w{5}|\d{5}");`

b) Create a regular expression to match a phone number in the form of (123) 456-7890.

ANS: `Regex regex = new Regex(@"\([1-9]\d{2}\)\s[1-9]\d{2}-\d{4}");`

Exercises and Answers

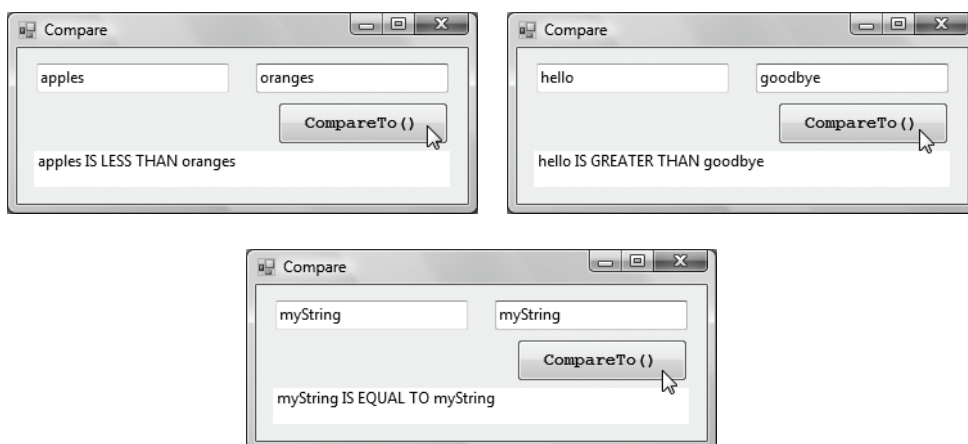
16.4 (*Comparing strings*) Write an application that uses string method `CompareTo` to compare two strings input by the user. Output whether the first string is less than, equal to or greater than the second.

ANS:

```

1  // Exercise 16.4: Compare.cs
2  // Using string method CompareTo.
3  using System;
4  using System.Windows.Forms;
5
6  public partial class CompareForm : Form
7  {
8      public CompareForm()
9      {
10         InitializeComponent();
11     } // end constructor
12
13     // compare the two strings
14     private void compareButton_Click( object sender, EventArgs e )
15     {
16         // invoke method CompareTo
17         int result =
18             firstTextBox.Text.CompareTo( secondTextBox.Text );
19
20         // display result
21         if ( result == 0 )
22             outputLabel.Text = firstTextBox.Text +
23                 " IS EQUAL TO " + secondTextBox.Text;
24         else if ( result > 0 )
25             outputLabel.Text = firstTextBox.Text +
26                 " IS GREATER THAN " + secondTextBox.Text;
27         else
28             outputLabel.Text = firstTextBox.Text +
29                 " IS LESS THAN " + secondTextBox.Text;
30
31         // clear text boxes
32         firstTextBox.Clear();
33         secondTextBox.Clear();
34         firstTextBox.Focus();
35     } // end method compareButton_Click
36 } // end class CompareForm

```



16.5 (*Random Sentences and Story Writer*) Write an application that uses random-number generation to create sentences. Use four arrays of strings, called `article`, `noun`, `verb` and `preposition`. Create a sentence by selecting a word at random from each array in the following order: `article`, `noun`, `verb`, `preposition`, `article`, `noun`. As each word is picked, concatenate it to the previous words in the sentence. The words should be separated by spaces. When the sentence is output, it should start with a capital letter and end with a period. The program should generate 10 sentences and output them to a text area.

The arrays should be filled as follows: The `article` array should contain the articles "the", "a", "one", "some" and "any"; the `noun` array should contain the nouns "boy", "girl", "dog", "town" and "car"; the `verb` array should contain the past-tense verbs "drove", "jumped", "ran", "walked" and "skipped"; and the `preposition` array should contain the prepositions "to", "from", "over", "under" and "on".

After the preceding program is written, modify the program to produce a short story consisting of several of these sentences. (How about the possibility of a random term-paper writer!)

ANS:

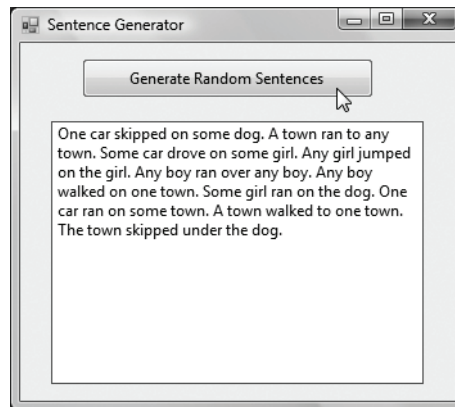
```
1 // Exercise 16.5: SentenceGenerator.cs
2 // Generating a random sentence.
3 using System;
4 using System.Windows.Forms;
5
6 namespace SentenceGenerator
7 {
8     public partial class SentenceGeneratorForm : Form
9     {
10         public SentenceGeneratorForm()
11         {
12             InitializeComponent();
13         } // end constructor
14
15         string[] articles = { "the", "a", "one", "some", "any" };
16         string[] nouns = { "boy", "girl", "dog", "town", "car" };
17         string[] verbs =
18             { "drove", "jumped", "ran", "walked", "skipped" };

```

```

19  string[] prepositions =
20      { "to", "from", "over", "under", "on" };
21
22  // generate ten sentences
23  private void generateButton_Click( object sender, EventArgs e )
24  {
25      // random object for selecting random words
26      Random random = new Random();
27      string sentence = String.Empty;
28
29      // clear output
30      outputTextBox.Text = "";
31
32      for ( int i = 0; i < 10; i++ )
33      {
34          // clear sentence
35          sentence = "";
36
37          // build random sentence
38          sentence += articles[ random.Next( articles.Length ) ] + " ";
39          sentence += nouns[ random.Next( nouns.Length ) ] + " ";
40          sentence += verbs[ random.Next( verbs.Length ) ] + " ";
41          sentence +=
42              prepositions[ random.Next( prepositions.Length ) ] + " ";
43          sentence += articles[ random.Next( articles.Length ) ] + " ";
44          sentence += nouns[ random.Next( nouns.Length ) ] + ".";
45
46          // capitalize first word
47          char firstLetter = Char.ToUpper( sentence[ 0 ] );
48          sentence = firstLetter + sentence.Substring( 1 );
49          outputTextBox.Text += sentence + " ";
50      } // end for
51  } // end method generateButton_Click
52  } // end class SentenceGeneratorForm
53  } // end namespace SentenceGenerator

```



16.6 (Pig Latin) Write an application that encodes English-language phrases into pig Latin. Pig Latin is a form of coded language often used for amusement. Many variations exist in the methods used to form pig Latin phrases. For simplicity, use the following algorithm:

To translate each English word into a pig Latin word, place the first letter of the English word at the end of the word and add the letters “ay.” Thus, the word “jump” becomes “umpjay,” the word “the” becomes “hetay” and the word “computer” becomes “omputercay.” Blanks between words remain blanks. Assume the following: The English phrase consists of words separated by blanks, there are no punctuation marks and all words have two or more letters. Enable the user to input a sentence. Use techniques discussed in this chapter to divide the sentence into separate words. Method `GetPigLatin` should translate a single word into pig Latin. Keep a running display of all the converted sentences in a text area.

ANS:

```

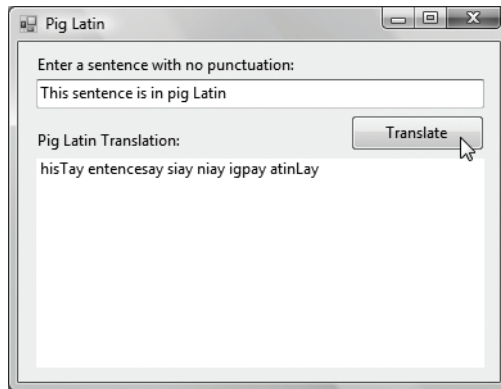
1  // Exercise 16.6: PigLatin.cs
2  // Using strings to implement a Pig Latin translator.
3  using System;
4  using System.Text;
5  using System.Windows.Forms;
6
7  public partial class PigLatinForm : Form
8  {
9      public PigLatinForm()
10     {
11         InitializeComponent();
12     } // end constructor
13
14     // translate from English to Pig Latin
15     private void translateButton_Click( object sender, EventArgs e )
16     {
17         // check for empty TextBox
18         if ( inputTextBox.Text == "" )
19         {
20             MessageBox.Show( "Please enter a sentence" );
21             inputTextBox.Focus();
22             return;
23         } // end if
24
25         // split the sentence into individual words
26         char[] separator = { ' ' };
27         string[] words = inputTextBox.Text.Split( separator );
28         string pigLatin = "";
29
30         // translate each word into pig latin
31         foreach ( string word in words )
32             pigLatin += GetPigLatin( word ) + " ";
33
34         // display the translation
35         outputTextBox.AppendText( pigLatin + "\r\n" );
36         inputTextBox.Clear();
37     } // end method translateButton_Click
38

```

```

39 // pressing enter is the same as clicking the Translate Button
40 private void inputTextBox_KeyDown( object sender, KeyEventArgs e )
41 {
42     // allow user to press enter in TextBox
43     if ( e.KeyCode == Keys.Enter )
44         translateButton_Click( sender, e );
45 } // end method inputTextBox_KeyDown
46
47 // translate the word
48 private string GetPigLatin( string word )
49 {
50     StringBuilder latin = new StringBuilder( word );
51     char firstLetter = latin[ 0 ];
52
53     // remove first letter and append it to the end
54     latin.Remove( 0, 1 );
55     latin.Append( firstLetter );
56
57     // add "ay" to the end of word
58     latin.Append( "ay" );
59     return latin.ToString();
60 } // end method GetPigLatin
61 } // end class PigLatinForm

```



16.7 (*All Possible Three-Letter Words from a Five-Letter Word*) Write a program that reads a five-letter word from the user and produces all possible three-letter combinations that can be derived from the letters of the five-letter word. For example, the three-letter words produced from the word “bathe” include the commonly used words “ate,” “bat,” “bet,” “tab,” “hat,” “the” and “tea,” and the 3-letter combinations “bth,” “eab,” etc.

ANS:

```

1 // Exercise 16.7: ThreeLetter.cs
2 // Finding all 3-letter words in a 5-letter word. There are
3 // 60 such combinations.
4 using System;
5 using System.Windows.Forms;

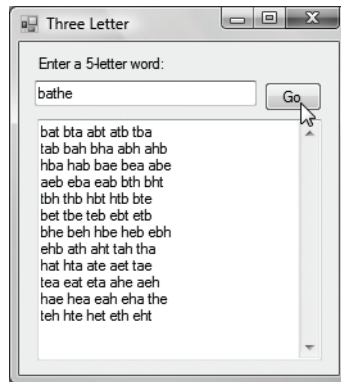
```

```
6
7 public partial class ThreeLetterForm : Form
8 {
9     public ThreeLetterForm()
10    {
11        InitializeComponent();
12    } // end constructor
13
14    // create all the three letter words
15    private void goButton_Click( object sender, EventArgs e )
16    {
17        if ( inputTextBox.Text.Length != 5 )
18        {
19            MessageBox.Show( "Enter a 5-letter word" );
20            inputTextBox.Focus();
21            return;
22        } // end if
23
24        string word = inputTextBox.Text;
25        string temp = "";
26
27        // clear outputTextBox
28        outputTextBox.Text = "";
29
30        // build a string containing all words
31        for ( int first = 0; first < 5; first++ )
32        {
33            for ( int second = first + 1; second < 5; second++ )
34            {
35                for ( int third = second + 1; third < 5; third++ )
36                {
37                    temp += word[ first ].ToString() +
38                        word[ second ] + word[ third ] + " ";
39                    temp += word[ first ].ToString() +
40                        word[ third ] + word[ second ] + " ";
41                    temp += word[ second ].ToString() +
42                        word[ first ] + word[ third ] + " ";
43                    temp += word[ second ].ToString() +
44                        word[ third ] + word[ first ] + " ";
45                    temp += word[ third ].ToString() +
46                        word[ first ] + word[ second ] + " ";
47                    temp += word[ third ].ToString() +
48                        word[ second ] + word[ first ] + " ";
49                } // end for
50            } // end for
51        } // end for
52
53        // split string into individual words
54        string[] words = temp.Split( ' ' );
55        int count = 0;
56
57        // display each three-letter word
58        foreach ( string w in words )
59        {
```

```

60         outputTextBox.AppendText( w + " " );
61         count++;
62
63         // start on a new line every five words
64         if ( count % 5 == 0 )
65             outputTextBox.AppendText( "\r\n" );
66     } // end foreach
67
68     inputTextBox.Clear();
69 } // end method goButton_Click
70
71 // pressing enter is the same as clicking the Go Button
72 private void inputTextBox_KeyDown( object sender, KeyEventArgs e )
73 {
74     // allow user to press enter in text box
75     ( e.KeyCode == Keys.Enter )
76     goButton_Click( sender, e );
77 } // end method inputTextBox_KeyDown
78 } // end class ThreeLetterForm

```



16.8 (*Capitalizing Words*) Write a program that uses regular expressions to convert the first letter of every word to uppercase. Have it do this for an arbitrary string input by the user.

ANS:

```

1  // Exercise 16.8: Capitalize.cs
2  // Capitalize the first letter of every word.
3  using System;
4  using System.Text.RegularExpressions;
5
6  class Capitalize
7  {
8      static void Main( string[] args )
9      {
10         string sentence; // input string
11         string output = string.Empty; // output string
12

```

```

13 // get the sentence from the user
14 Console.WriteLine( "Please enter a sentence:" );
15 sentence = Console.ReadLine();
16
17 // split the sentence into words
18 string[] words = Regex.Split( sentence, @"\s" );
19
20 // change the first letter of each word to uppercase
21 foreach ( var word in words )
22 {
23     output += Char.ToUpper( word[ 0 ] ) +
24         word.Substring( 1 ) + " ";
25 } // end foreach
26
27 Console.WriteLine( output ); // display the output string
28 } // end Main
29 } // end class Capitalize

```

```

Please enter a sentence:
welcome to visual c# 2010
Welcome To Visual C# 2010

```

16.9 (*Counting Characters of Different Types*) Use a regular expression to count the number of digits, characters and whitespace characters in a string. [*Hint: Regex function Matches returns an IEnumerable, which has a Count extension method.*]

ANS:

```

1 // Exercise 16.9: CharacterCount.cs
2 // Count the number of digits, word characters and whitespace characters.
3 using System;
4 using System.Text.RegularExpressions;
5
6 class CharacterCount
7 {
8     static void Main( string[] args )
9     {
10         int digits, words, whitespace; // character counts
11
12         // get sentence from the user
13         Console.WriteLine( "Please enter a sentence:" );
14         string sentence = Console.ReadLine();
15
16         // count the number of digits
17         digits = Regex.Matches( sentence, @"\d" ).Count;
18
19         // count the number of word characters
20         words = Regex.Matches( sentence, @"\w" ).Count;
21
22         // count the number of whitespace characters
23         whitespace = Regex.Matches( sentence, @"\s" ).Count;
24

```

```

25         // display the results
26         Console.WriteLine( "\nThere are {0} digits, " +
27             "{1} word characters and {2} whitespace characters.",
28             digits, words, whitespace );
29     } // end Main
30 } // end class CharacterCount

```

Please enter a sentence:
Welcome to Visual C# 2010 HTP

There are 4 digits, 23 word characters and 5 whitespace characters.

16.10 (Validating Numbers) Write a regular expression that will search a string and match a valid number. A number can have any number of digits, but it can have only digits and a decimal point. The decimal point is optional, but if it appears in the number, there must be only one, and it must have digits on its left and its right. There should be whitespace or a beginning- or end-of-line character on either side of a valid number. Negative numbers are preceded by a minus sign.

ANS:

```

1  // Exercise 16.10: NumberSearch.cs
2  // Use regular expressions to search a string for a number.
3  using System;
4  using System.Text.RegularExpressions;
5
6  class NumberSearch
7  {
8      static void Main( string[] args )
9      {
10         Regex expression = new Regex( @"(\s|^)-?\d+(\.\d+)?(\s|$)" );
11
12         // get input string from the user
13         Console.WriteLine( "Enter a string with a number:" );
14         string sentence = Console.ReadLine();
15
16         // check if the sentence contained any valid numbers
17         if ( !expression.IsMatch( sentence ) )
18             Console.WriteLine( "No numbers found in the string." );
19         else
20         {
21             Console.WriteLine( "\nNumber(s) found in the string:" );
22
23             // display all matches to the regular expression
24             foreach ( var myMatch in expression.Matches( sentence ) )
25                 Console.Write( "{0} ", myMatch );
26
27             Console.WriteLine(); // new line after output
28         } // end else
29     } // end Main
30 } // end class NumberSearch

```

```
Enter a string with a number:
2 times 5 is 10

Number(s) found in the string:
2 5 10
```

16.11 (*Removing Excess Space Between Words*) Write a program that asks the user to enter a sentence and uses a regular expression to check whether the sentence contains more than one space between words. If so, the program should remove the extra spaces. For example, "Hello World" should be "Hello World".

ANS:

```
1 // Exercise 16.11: RemoveSpaces.cs
2 // Remove extra spaces from a sentence.
3 using System;
4 using System.Text.RegularExpressions;
5
6 class RemoveSpaces
7 {
8     static void Main( string[] args )
9     {
10         // get the sentence from the user
11         Console.WriteLine( "Please enter a sentence:" );
12         string sentence = Console.ReadLine();
13
14         // match two or more consecutive whitespace characters
15         Regex spaces = new Regex( @"\s{2,}" );
16
17         // check for extra spaces
18         if ( !spaces.IsMatch( sentence ) )
19             Console.WriteLine( "\nNo extra spaces!" );
20         else
21         {
22             // replace any occurrences of multiple spaces with a single space
23             sentence = spaces.Replace( sentence, " " );
24
25             // display the corrected sentence
26             Console.WriteLine( "\nExtra spaces have been removed" );
27             Console.WriteLine( sentence );
28         } // end else
29     } // end Main
30 } // end class RemoveSpaces
```

```
Please enter a sentence:
Hello World

Extra spaces have been removed
Hello World
```

Files and Streams

17

I can only assume that a “Do Not File” document is filed in a “Do Not File” file.

—Senator Frank Church
Senate Intelligence Subcommittee
Hearing, 1975

Consciousness ... does not appear to itself chopped up in bits. ... A “river” or a “stream” are the metaphors by which it is most naturally described.

—William James

I read part of it all the way through.

—Samuel Goldwyn

Objectives

In this chapter you’ll learn:

- To create, read, write and update files.
- To use classes `File` and `Directory` to obtain information about files and directories on your computer.
- To use LINQ to search through directories.
- To become familiar with sequential-access file processing.
- To use classes `FileStream`, `StreamReader` and `StreamWriter` to read text from and write text to files.
- To use classes `FileStream` and `BinaryFormatter` to read objects from and write objects to files.

Self-Review Exercises

- 17.1** State whether each of the following is *true* or *false*. If *false*, explain why.
- a) Creating instances of classes `File` and `Directory` is impossible.
ANS: True.
 - b) Typically, a sequential file stores records in order by the record-key field.
ANS: True.
 - c) Class `StreamReader` inherits from class `Stream`.
ANS: False. Class `StreamReader` inherits from class `TextReader`.
 - d) Any class can be serialized to a file.
ANS: False. Only classes that implement interface `ISerializable` or are declared with the `Serializable` attribute can be serialized.
 - e) Method `Seek` of class `FileStream` always seeks relative to the beginning of a file.
ANS: False. It seeks relative to the `SeekOrigin` enumeration member that is passed as one of the arguments.
 - f) Classes `StreamReader` and `StreamWriter` are used with sequential-access files.
ANS: True.
 - g) Instantiating objects of type `Stream` is impossible.
ANS: True.
- 17.2** Fill in the blanks in each of the following:
- a) Ultimately, all data items processed by a computer are reduced to combinations of _____ and _____.
ANS: 0s, 1s.
 - b) The smallest data item a computer can process is called a _____.
ANS: bit.
 - c) A _____ is a group of related records.
ANS: file.
 - d) Digits, letters and special symbols are collectively referred to as _____.
ANS: characters.
 - e) A group of related files is called a _____.
ANS: database.
 - f) `StreamReader` method _____ reads a line of text from a file.
ANS: `ReadLine`.
 - g) `StreamWriter` method _____ writes a line of text to a file.
ANS: `WriteLine`.
 - h) Method `Serialize` of class `BinaryFormatter` takes a(n) _____ and a(n) _____ as arguments.
ANS: `Stream`, object.
 - i) The _____ namespace contains most of C#'s file-processing classes.
ANS: `System.IO`.
 - j) The _____ namespace contains the `BinaryFormatter` class.
ANS: `System.Runtime.Serialization.Formatters.Binary`.

Exercises

17.3 (*File of Student Grades*) Create a program that stores student grades in a text file. The file should contain the name, ID number, class taken and grade of every student. Allow the user to load a grade file and display its contents in a read-only `TextBox`. The entries should be displayed in the following format:

```
LastName, FirstName: ID# Class Grade
```

We list some sample data below:

```
Jones, Bob: 1 "Introduction to Computer Science" "A-"
Johnson, Sarah: 2 "Data Structures" "B+"
Smith, Sam: 3 "Data Structures" "C"
```

ANS:

```

1 // Ex. 17.3: GradesForm.cs
2 // Writing student information to a file sequentially.
3 using System;
4 using System.Windows.Forms;
5 using System.IO;
6
7 namespace Grades
8 {
9     public partial class GradesForm : Form
10    {
11        private StreamWriter output;
12        private StreamReader input;
13        private bool saved = false;
14
15        // default constructor
16        public GradesForm()
17        {
18            InitializeComponent();
19        } // end constructor
20
21        // handles TextChanged event for all the input TextBoxes
22        private void AllTextBox_TextChanged( object sender, EventArgs e )
23        {
24
25            EnableEnterButton(); // check if enter Button should be enabled
26        } // end AllTextBox_TextChanged
27
28        // handles click event for save as Button
29        private void saveAsButton_Click( object sender, EventArgs e )
30        {
31            // show save file dialog
32            DialogResult result; // result of SaveFileDialog
33            string fileName;
34
35            using ( SaveFileDialog chooser = new SaveFileDialog() )
36            {
37                result = chooser.ShowDialog();
38                fileName = chooser.FileName; // name of file to save data
39            } // end using
40
41            // ensure that user clicked OK
42            if ( result == DialogResult.OK )
43            {
44                if ( fileName == string.Empty )
45                    MessageBox.Show( "You entered an invalid file name.",
46                                    "Invalid File name", MessageBoxButtons.OK,
47                                    MessageBoxIcon.Information );
48            }
49        }
50    }
51 }

```

```
48         else
49         {
50             output = new StreamWriter( fileName );
51             saveAsButton.Enabled = false;
52             saved = true;
53         } // end else
54
55         // check if enter Button should be enabled
56         EnableEnterButton();
57     } // end if
58 } // end method saveAsButton_Click
59
60 // enable enterButton when all text boxes are filled
61 // and a file is open
62 private void EnableEnterButton()
63 {
64     if ( lastTextBox.Text != string.Empty &&
65         firstTextBox.Text != string.Empty &&
66         idTextBox.Text != string.Empty &&
67         classTextBox.Text != string.Empty &&
68         gradeTextBox.Text != string.Empty && saved )
69         enterButton.Enabled = true;
70     else
71         enterButton.Enabled = false;
72 } // end method EnableEnterButton
73
74 private void enterButton_Click( object sender, EventArgs e )
75 {
76     // retrieve user input
77     string last = lastTextBox.Text;
78     string first = firstTextBox.Text;
79     string id = idTextBox.Text;
80     string className = classTextBox.Text;
81     string grade = gradeTextBox.Text;
82
83     output.WriteLine( last + "\t" + first + "\t" + id +
84         "\t" + className + "\t" + grade );
85
86     statusLabel.Text = "Entry saved"; // update display
87
88     // clear text boxes
89     lastTextBox.Clear();
90     firstTextBox.Clear();
91     idTextBox.Clear();
92     classTextBox.Clear();
93     gradeTextBox.Clear();
94 } // end method enterButton_Click
95
96 private void loadButton_Click( object sender, EventArgs e )
97 {
98     // make sure StreamWriter object is closed and equals null
99     if ( output != null )
100     {
101         output.Close();
```

```

102         output = null;
103         statusLabel.Text = "Closing file";
104         saveAsButton.Enabled = true;
105         saved = false;
106     } // end if
107
108     // show load file dialog
109     DialogResult result;
110     string fileName;
111
112     using ( OpenFileDialog chooser = new OpenFileDialog() )
113     {
114         result = chooser.ShowDialog();
115         fileName = chooser.FileName;
116     } // end using
117
118     // make sure that user didn't cancel
119     if ( result == DialogResult.OK )
120     {
121         if ( fileName == string.Empty )
122             MessageBox.Show( "Invalid File Name", "Invalid File name",
123                             MessageBoxButtons.OK, MessageBoxIcon.Information );
124         else
125         {
126             input = new StreamReader( fileName );
127             string entry = input.ReadLine();
128
129             gradesTextBox.Clear(); // clear text box
130
131             // continue reading in entries until end of file
132             while ( entry != null )
133             {
134                 // otherwise, add to textbox
135                 gradesTextBox.AppendText( FormatEntry( entry ) +
136                                         "\r\n" );
137
138                 entry = input.ReadLine();
139             } // end while
140
141             statusLabel.Text = "File loaded";
142         } // end else
143     }
144 } // end method loadButton_Click
145
146 // set format to display data in TextBox
147 private string FormatEntry( string fromFile )
148 {
149     // split string from file
150     char[] splitters = { '\t' };
151     string[] data = fromFile.Split( splitters );
152
153     // format results for display
154     string result = data[ 0 ] + ", " + data[ 1 ] +
155         ":\t" + data[ 2 ] + "\t" + data[ 3 ] + "\t" + data[ 4 ];

```

```

156
157         return result;
158     } // end method FormatEntry
159 } // end class GradesForm
160 } // end namespace Grades

```

Grades

Last Name: Jones

First Name: Bob

ID#: 1

Class: Intro to Programming

Grade: B

Save As Load Enter

Status Label

Grades

Grades

Last Name: Johnson

First Name: Sarah

ID#: 2

Class: Data Structures

Grade: B+

Save As Load Enter

Entry saved

Grades

Grades

Last Name: Smith

First Name: Sam

ID#: 3

Class: Data Structures

Grade: C

Save As Load Enter

Entry saved

Grades

Grades

Last Name:

First Name:

ID#

Class

Grade

Save As Load Enter

File loaded

Grades

Jones, Bob:	1	Intro to Programming
B		
Johnson, Sarah:	2	Data Structures
Smith, Sam:	3	Data Structures
		B+
		C

17.4 (*Serializing and Deserializing*) Modify the previous program to use objects of a class that can be serialized to and deserialized from a file.

ANS:

```

1 // Ex. 17.4: ClassGrades.cs
2 // Serializable class that stores a student's grade
3 using System;
4
5 [ Serializable ]
6 class ClassGrade
7 {
8     public string Last { get; set; }
9     public string First { get; set; }
10    public string Id { get; set; }
11    public string Class { get; set; }
12    public string Grade { get; set; }
13
14    // default constructor
15    public ClassGrade()
16    {
17    } // end constructor
18
19    // overloaded constructor sets members to parameter values
20    public ClassGrade( string lastName, string firstName,
21        string id, string className, string grade )
22    {
23        Last = lastName;
24        First = firstName;
25        Id = id;
26        Class = className;
27        Grade = grade;
28    } // end constructor
29 } // end class ClassGrade

```

```

1 // Ex. 17.4: GradeForms.cs
2 // Writing ClassGrade objects to a file.
3 using System;
4 using System.Windows.Forms;
5 using System.IO;
6 using System.Runtime.Serialization;
7 using System.Runtime.Serialization.Formatters.Binary;
8
9 namespace Grades
10 {
11     public partial class GradesForm : Form
12     {
13         private FileStream fileInOut;
14         private bool saved = false;
15         private BinaryFormatter formatter;
16
17         // default constructor
18         public GradesForm()
19         {

```

```

20         InitializeComponent();
21
22         formatter = new BinaryFormatter(); // create new BinaryFormatter
23     } // end constructor
24
25     // handles TextChanged event for all the input TextBoxes
26     private void AllTextBox_TextChanged( object sender, EventArgs e )
27     {
28         EnableEnterButton(); // check if enter Button should be enabled
29     } // end method AllTextBox_TextChanged
30
31     // handles click event for save as Button
32     private void saveAsButton_Click( object sender, EventArgs e )
33     {
34         // make sure FileStream is not in use
35         if ( fileInOut != null )
36         {
37             fileInOut.Close();
38             fileInOut = null;
39             statusLabel.Text = "Closing file";
40         } // end if
41
42         // show save file dialog
43         DialogResult result; // result of OpenFileDialog
44         string fileName; // name of file containing data
45
46         using ( SaveFileDialog chooser = new SaveFileDialog() )
47         {
48             result = chooser.ShowDialog();
49             fileName = chooser.FileName;
50         } // end using
51
52         // ensure that user clicked OK
53         if ( result == DialogResult.OK )
54         {
55             if ( fileName == string.Empty )
56                 MessageBox.Show( "Invalid File Name", "Invalid File name",
57                                 MessageBoxButtons.OK, MessageBoxIcon.Information );
58             else
59             {
60                 fileInOut = new FileStream( fileName,
61                                           FileMode.OpenOrCreate );
62
63                 saveAsButton.Enabled = false;
64                 saved = true;
65             } // end else
66
67             // check if enter Button should be enabled
68             EnableEnterButton();
69         } // end if
70     } // end method saveAsButton_Click
71

```

```

72 // enable enterButton when all text boxes are filled
73 // and a file is open
74 private void EnableEnterButton()
75 {
76     if ( lastTextBox.Text != string.Empty &&
77         firstTextBox.Text != string.Empty &&
78         idTextBox.Text != string.Empty &&
79         classTextBox.Text != string.Empty &&
80         gradeTextBox.Text != string.Empty && saved )
81         enterButton.Enabled = true;
82     else
83         enterButton.Enabled = false;
84 } // end method EnableEnterButton
85
86 // handles click event for enter Button
87 private void enterButton_Click( object sender, EventArgs e )
88 {
89     // retrieve user input
90     string last = lastTextBox.Text;
91     string first = firstTextBox.Text;
92     string id = idTextBox.Text;
93     string className = classTextBox.Text;
94     string grade = gradeTextBox.Text;
95
96     ClassGrade entry = new ClassGrade( last, first, id,
97         className, grade );
98
99     // write object to file sequentially
100    formatter.Serialize( fileInOut, entry );
101
102    // update display
103    statusLabel.Text = "Entry saved";
104
105    // clear text boxes
106    lastTextBox.Clear();
107    firstTextBox.Clear();
108    idTextBox.Clear();
109    classTextBox.Clear();
110    gradeTextBox.Clear();
111 } // end method enterButton_Click
112
113 // handles click event for load Button
114 private void loadButton_Click( object sender, EventArgs e )
115 {
116     // make sure FileStream is not in use
117     if ( fileInOut != null )
118     {
119         fileInOut.Close();
120         fileInOut = null;
121         statusLabel.Text = "Closing file";
122         saveAsButton.Enabled = true;
123         saved = false;
124     } // end if
125

```

```

126         // create and show dialog box enabling user to open file
127         DialogResult result; // result of OpenFileDialog
128         string fileName; // name of file containing data
129
130         using ( OpenFileDialog fileChooser = new OpenFileDialog() )
131         {
132             result = fileChooser.ShowDialog();
133             fileName = fileChooser.FileName; // get specified name
134         }
135
136         // ensure that user clicked "OK"
137         if ( result == DialogResult.OK )
138         {
139
140             if ( fileName == string.Empty )
141                 MessageBox.Show( "Invalid File Name", "Invalid File name",
142                                 MessageBoxButtons.OK, MessageBoxIcon.Information );
143             else
144             {
145                 // prepare to read object
146                 fileInOut = new FileStream( fileName,
147                                           FileMode.Open, FileAccess.Read );
148
149                 gradesTextBox.Clear();
150
151                 try
152                 {
153                     while ( true ) // loop until there are no more records
154                     {
155                         ClassGrade entry =
156                             ( ClassGrade ) formatter.Deserialize( fileInOut );
157                         gradesTextBox.AppendText( FormatEntry( entry ) +
158                                                  "\r\n" );
159                     } // end while
160                 } // end try
161                 catch ( SerializationException )
162                 {
163                     statusLabel.Text = "File loaded";
164                 } // end catch
165             } // end else
166         } // end if
167     } // end method loadButton_Click
168
169     // set format to display properties of ClassGrade in TextBox
170     private string FormatEntry( ClassGrade fromFile )
171     {
172         // format results for display:
173         string result = fromFile.Last.Trim() + ", " +
174             fromFile.First.Trim() + ":\t" + fromFile.Id.Trim() + "\t" +
175             fromFile.Class.Trim() + "\t" + fromFile.Grade.Trim();
176
177         return result;
178     } // end method FormatEntry
179 } // end class GradesForm
180 } // end namespace Grades

```

The figure shows four sequential screenshots of a Java Swing window titled "Grades".

- Screenshot 1 (Top Left):** The form contains the following data:

Last Name:	Jones
First Name:	Bob
ID#	1
Class	Intro to Programming
Grade	B

 The "Enter" button is highlighted. Below the form is a label "Status Label" and an empty "Grades" list box.
- Screenshot 2 (Top Right):** The form contains:

Last Name:	Johnson
First Name:	Sarah
ID#	2
Class	Data Structures
Grade	B+

 The "Enter" button is highlighted. The status label now says "Entry saved".
- Screenshot 3 (Bottom Left):** The form contains:

Last Name:	Smith
First Name:	Sam
ID#	3
Class	Data Structures
Grade	C

 The "Enter" button is highlighted. The status label says "Entry saved".
- Screenshot 4 (Bottom Right):** The form fields are empty. The "Load" button is highlighted. The status label says "File loaded". The "Grades" list box now contains the following text:

Jones, Bob:	1	Intro to Programming
B		
Johnson, Sarah:	2	Data Structures
B+		
Smith, Sam:	3	Data Structures
C		

17.5 (*Extending `StreamReader` and `StreamWriter`*) Extend classes `StreamReader` and `StreamWriter`. Make the class that derives from `StreamReader` have methods `ReadInteger`, `ReadBoolean` and `ReadString`. Make the class that derives from `StreamWriter` have methods `WriteInteger`, `WriteBoolean` and `WriteString`. Think about how to design the writing methods so that the reading

methods will be able to read what was written. Design `WriteInteger` and `WriteBoolean` to write strings of uniform size so that `ReadInteger` and `ReadBoolean` can read those values accurately. Make sure `ReadString` and `WriteString` use the same character(s) to separate strings.

ANS:

```

1  // Ex. 17.5: MyReader.cs
2  // Defining an extended StreamReader.
3  using System;
4  using System.IO;
5
6  class MyReader : StreamReader
7  {
8      // default constructor
9      public MyReader( string fileName )
10         : base( fileName )
11     {
12     } // end constructor
13
14     // read in integer from file
15     public int ReadInteger()
16     {
17         // for reading an integer
18         char[] buffer = new char[ 10 ];
19         int input = 0;
20         string toRead = string.Empty;
21
22         // read 10 characters representing an integer from the file
23         Read( buffer, 0, 10 );
24
25         toRead = new string( buffer );
26
27         try
28         {
29             input = Int32.Parse( toRead );
30         } // end try
31         catch ( Exception )
32         {
33             return 0;
34         } // end catch
35
36         return input;
37     } // end method ReadInteger
38
39     // read in boolean from file
40     public bool ReadBoolean()
41     {
42         // for reading a boolean
43         char[] buffer = new char[ 5 ];
44         string toRead = string.Empty;
45
46         // read either 'true' or 'false'
47         Read( buffer, 0, 5 );
48

```

```

49     toRead = new string( buffer );
50
51     return toRead.Trim() == "true";
52 } // end method ReadBoolean
53
54 // read in string from file
55 public string ReadString()
56 {
57     // for reading a string
58     char[] buffer = new char[ 1 ];
59     string input = string.Empty;
60
61     // each string is separated by "ENDOFSTRING";
62     // read in one character at a time until termination
63     // string is reached
64     while ( input.LastIndexOf( "ENDOFSTRING" ) == -1 )
65     {
66         Read( buffer, 0, 1 );
67         input += buffer[ 0 ];
68     } // end while
69
70     // trim the termination string off the read string
71     return input.Substring( 0, input.LastIndexOf( "ENDOFSTRING" ) );
72 } // end method ReadString
73 } // end class MyReader

```

```

1  // Ex. 17.5: MyWriter.cs
2  // Defining an extended StreamWriter.
3  using System;
4  using System.IO;
5
6  class MyWriter : StreamWriter
7  {
8      // constructor for MyWriter object
9      public MyWriter( string fileName )
10         : base( fileName )
11     {
12     } // end constructor
13
14     // write integer to file
15     public void WriteInteger( int output )
16     {
17         // pad the integer with zeros
18         string toWrite = output.ToString().PadLeft( 10, '0' );
19
20         Write( toWrite ); // write to file
21     } // end method WriteInteger
22
23     // write boolean to file
24     public void WriteBoolean( bool output )
25     {

```

```

26         // a written boolean is 5 characters long
27         if ( output )
28             Write( "true " );
29         else
30             Write( "false" );
31     } // end method WriteBoolean
32
33     // write string to file
34     public void WriteString( string output )
35     {
36         // each string ends with "ENDOFSTRING"
37         Write( output + "ENDOFSTRING" );
38     } // end method WriteString
39 } // end class MyWriter

```

```

1  // Ex. 17.5: Extend.cs
2  // Overloading StreamReader and StreamWriter.
3  using System;
4  using System.IO;
5
6  class Extend
7  {
8      public static void Main( string[] args )
9      {
10         // create a file for testing
11         FileStream s = File.Create( "test.txt" );
12         s.Close();
13
14         // create MyWriter object
15         MyWriter write = new MyWriter( "test.txt" );
16
17         // write some integers to a file
18         Console.WriteLine(
19             "Writing the following integers: 4, 38764, 20948" );
20         write.WriteInteger( 4 );
21         write.WriteInteger( 38764 );
22         write.WriteInteger( 20948 );
23
24         // write some booleans
25         Console.WriteLine(
26             "Writing the following booleans: true, false" );
27         write.WriteBoolean( true );
28         write.WriteBoolean( false );
29
30         string day = "Have a nice day";
31         string year = "Happy new year!";
32
33         // write some strings
34         Console.WriteLine( "Writing the following strings: \"\"
35             + day + "\", \"" + year + "\"\" );
36         write.WriteString( day );
37         write.WriteString( year );
38

```

```

39     write.Close(); // close the file
40
41     MyReader read = new MyReader( "test.txt" );
42
43     // read integers back from the file
44     int firstInt = read.ReadInteger();
45     int secondInt = read.ReadInteger();
46     int thirdInt = read.ReadInteger();
47
48     Console.WriteLine( "\nRead the following integers: " +
49         "{0}, {1}, {2}", firstInt, secondInt, thirdInt );
50
51     // read booleans
52     bool firstBool = read.ReadBoolean();
53     bool secondBool = read.ReadBoolean();
54
55     Console.WriteLine( "Read the following booleans: " +
56         "{0}, {1}", firstBool, secondBool );
57
58     // read strings
59     string firstString = read.ReadString();
60     string secondString = read.ReadString();
61
62     Console.WriteLine( "Read the following strings: " +
63         "\"{0}\", \"{1}\"", firstString, secondString );
64 } // end Main
65 } // end class Extend

```

```

Writing the following integers: 4, 38764, 20948
Writing the following booleans: true, false
Writing the following strings: "Have a nice day", "Happy new year!"

Read the following integers: 4, 38764, 20948
Read the following booleans: True, False
Read the following strings: "Have a nice day", "Happy new year!"

```

17.6 (*Reading and Writing Account Information*) Create a program that combines the ideas of Fig. 19.9 and Fig. 19.11 to allow a user to write records to and read records from a file. Add an extra field of type `bool` to the record to indicate whether the account has overdraft protection.

ANS:

```

1  // Ex. 17.6: Record2.cs
2  // Extends the Record class of BankLibrary and contains
3  // information about the customer's overdraft protection.
4  using System;
5  using BankLibrary;
6
7  [ Serializable ]
8  public class Record2 : RecordSerializable
9  {

```

```

10 // automatic property for bool OverdraftProtection
11 public bool OverdraftProtection { get; set; }
12
13 // default constructor
14 public Record2() : base()
15 {
16     OverdraftProtection = false;
17 } // end constructor
18
19 // overloaded constructor
20 public Record2( int account, string firstName,
21     string lastName, decimal balance, bool overdraft )
22     : base( account, firstName, lastName, balance )
23 {
24     OverdraftProtection = overdraft;
25 } // end constructor
26 } // end class Record2

```

```

1 // Ex. 17.6: WriteReadFileForm.cs
2 // Writing to and reading from a sequential-access file.
3 using System;
4 using System.Windows.Forms;
5 using System.IO;
6 using System.Runtime.Serialization.Formatters.Binary;
7 using System.Runtime.Serialization;
8 using BankLibrary;
9
10 namespace WriteReadFile
11 {
12     public partial class WriteReadFileForm : BankUIForm
13     {
14         // stream through which serializable data are read from file
15         private FileStream theFile;
16
17         // object for deserializing Record2 in binary format
18         private BinaryFormatter formatter = new BinaryFormatter();
19
20         // default constructor
21         public WriteReadFileForm()
22         {
23             InitializeComponent();
24         } // end constructor
25
26         // handles click event for enter Button
27         private void enterButton_Click( object sender, System.EventArgs e )
28         {
29             // store TextBox values string array
30             string[] values = GetTextBoxValues();
31
32             // Record2 containing TextBox values to serialize
33             Record2 record = new Record2();
34

```

```

35         // determine whether TextBox account field is empty
36         if ( values[ ( int ) TextBoxIndices.ACCOUNT ] != string.Empty )
37         {
38             // store TextBox values in Record2 and serialize Record2
39             try
40             {
41                 // get account number value from TextBox
42                 int accountNumber = Int32.Parse(
43                     values[ ( int ) TextBoxIndices.ACCOUNT ] );
44
45                 // determine whether accountNumber is valid
46                 if ( accountNumber > 0 )
47                 {
48                     // store TextBox fields in Record2
49                     record.Account = accountNumber;
50                     record.FirstName = values[ ( int )
51                         TextBoxIndices.FIRST ];
52                     record.LastName = values[ ( int ) TextBoxIndices.LAST ];
53                     record.Balance = Decimal.Parse( values[
54                         ( int ) TextBoxIndices.BALANCE ] );
55                     record.OverdraftProtection = overdraftCheckBox.Checked;
56
57                     // write Record2 to FileStream (serialize object)
58                     formatter.Serialize( theFile, record );
59                 } // end if
60                 else
61                 {
62                     // notify user if invalid account number
63                     MessageBox.Show( "Invalid Account Number", "Error",
64                         MessageBoxButtons.OK, MessageBoxIcon.Error );
65                 } // end else
66             } // end try
67             // notify user if error occurs in serialization
68             catch ( SerializationException )
69             {
70                 MessageBox.Show( "Error Writing to File", "Error",
71                     MessageBoxButtons.OK, MessageBoxIcon.Error );
72             } // end catch
73             // notify user if error occurs regarding parameter format
74             catch ( FormatException )
75             {
76                 MessageBox.Show( "Invalid Format", "Error",
77                     MessageBoxButtons.OK, MessageBoxIcon.Error );
78             } // end catch
79         } // end if
80
81         ClearTextBoxes(); // clear TextBox values
82     } // end method enterButton_Click
83
84     // invoked when user clicks Save button
85     private void saveButton_Click( object sender, System.EventArgs e )
86     {
87         // create and show dialog box enabling user to save file
88         DialogResult result; // result of SaveFileDialog

```

```

89         string fileName; // name of file containing data
90
91         using ( SaveFileDialog fileChooser = new SaveFileDialog() )
92         {
93             // allow user to create file
94             fileChooser.CheckFileExists = false;
95             result = fileChooser.ShowDialog();
96             fileName = fileChooser.FileName; // name of file to save data
97         } // end using
98
99         // ensure that user clicked "OK"
100        if ( result == DialogResult.OK )
101        {
102
103            // show error if user specified invalid file
104            if ( fileName == string.Empty )
105                MessageBox.Show( "Invalid File Name", "Error",
106                                MessageBoxButtons.OK, MessageBoxIcon.Error );
107            else
108            {
109                // save file via FileStream if user specified valid file
110                try
111                {
112                    if ( theFile != null )
113                        theFile.Close();
114
115                    // open file with write access
116                    theFile = new FileStream( fileName,
117                                            FileMode.OpenOrCreate, FileAccess.Write );
118
119                    // disable Save button and enable Enter button
120                    saveButton.Enabled = false;
121                    enterButton.Enabled = true;
122                    nextButton.Enabled = false;
123                } // end try
124                // handle exception if file does not exist
125                catch ( FileNotFoundException )
126                {
127                    // notify user if file does not exist
128                    MessageBox.Show( "File Does Not Exist", "Error",
129                                    MessageBoxButtons.OK, MessageBoxIcon.Error );
130                } // end catch
131            } // end else
132        } // end if
133    } // end method saveButton_Click
134
135    // invoked when user clicks Open button
136    private void loadButton_Click(
137        object sender, System.EventArgs e )
138    {
139        // create and show dialog box enabling user to open file
140        DialogResult result; // result of OpenFileDialog
141        string fileName; // name of file containing data
142

```

```

143     using ( OpenFileDialog fileChooser = new OpenFileDialog() )
144     {
145         result = fileChooser.ShowDialog();
146         fileName = fileChooser.FileName; // get specified name
147     } // end using
148
149     // ensure that user clicked "OK"
150     if ( result == DialogResult.OK )
151     {
152         ClearTextBoxes();
153
154         // show error if user specified invalid file
155         if ( fileName == string.Empty )
156             MessageBox.Show( "Invalid File Name", "Error",
157                             MessageBoxButtons.OK, MessageBoxIcon.Error );
158         else
159         {
160             if ( theFile != null )
161                 theFile.Close();
162
163             // create FileStream to obtain read access to file
164             theFile = new FileStream( fileName, FileMode.Open,
165                                     FileAccess.Read );
166
167             // enable next record button
168             nextButton.Enabled = true;
169             enterButton.Enabled = false;
170             loadButton.Enabled = false;
171         } // end else
172     } // end if
173 } // end method loadButton_Click
174
175 // invoked when user clicks Next button
176 private void nextButton_Click(
177     object sender, System.EventArgs e )
178 {
179     // deserialize Record2 and store data in TextBoxes
180     try
181     {
182         // get next Record2 available in file
183         Record2 record =
184             ( Record2 ) formatter.Deserialize( theFile );
185
186         // store Record2 values in temporary string array
187         string[] values = new string[] {
188             record.Account.ToString(),
189             record.FirstName.ToString(),
190             record.LastName.ToString(),
191             record.Balance.ToString() };
192
193         // copy string array values to TextBox values
194         SetTextBoxValues( values );
195
196         overdraftCheckBox.Checked = record.OverdraftProtection;
197     } // end try

```

```

198         // handle exception when no Record2s in file
199     catch ( SerializationException )
200     {
201         // close FileStream if no Record2s in file
202         theFile.Close();
203
204         // enable Load button
205         loadButton.Enabled = true;
206
207         // disable Next button
208         nextButton.Enabled = false;
209
210         ClearTextBoxes();
211
212         // notify user if no records in file
213         MessageBox.Show( "No more records in file", string.Empty,
214             MessageBoxButtons.OK, MessageBoxIcon.Information );
215     } // end catch
216 } // end method nextButton_Click
217 } // end class WriteReadFileForm
218 } // end namespace WriteReadFile

```

Serializing Records

Account: 100

First Name: Nancy

Last Name: Brown

Balance: -25.54

☐ Overdraft Protection

Save Load Enter Next

Serializing Records

Account: 200

First Name: Stacey

Last Name: Dunn

Balance: 314.33

☒ Overdraft Protection

Save Load Enter Next

Serializing Records

Account: 300

First Name: Doug

Last Name: Decker

Balance: 0.00

☒ Overdraft Protection

Save Load Enter Next

Serializing Records

Account: 100

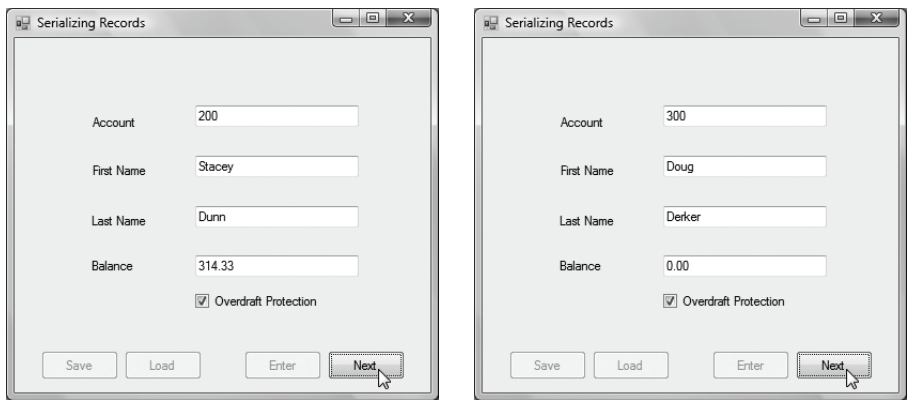
First Name: Nancy

Last Name: Brown

Balance: -25.54

☐ Overdraft Protection

Save Load Enter Next



17.7 (Telephone-Number Word Generator) Standard telephone keypads contain the digits zero through nine. The numbers two through nine each have three letters associated with them (Fig. 17.16). Many people find it difficult to memorize phone numbers, so they use the correspondence between digits and letters to develop seven-letter words that correspond to their phone numbers. For example, a person whose telephone number is 686-2377 might use the correspondence indicated in Fig. 17.16 to develop the seven-letter word “NUMBERS.” Every seven-letter word corresponds to exactly one seven-digit telephone number. A restaurant wishing to increase its takeout business could surely do so with the number 825-3688 (i.e., “TAKEOUT”).

Digit	Letter
2	A B C
3	D E F
4	G H I
5	J K L
6	M N O
7	P R S
8	T U V
9	W X Y Z

Fig. 17.16 | Letters that correspond to the numbers on a telephone keypad .

Every seven-letter phone number corresponds to many different seven-letter words. Unfortunately, most of these words represent unrecognizable juxtapositions of letters. It’s possible, however, that the owner of a barbershop would be pleased to know that the shop’s telephone number, 424-7288, corresponds to “HAIRCUT.” The owner of a liquor store would no doubt be delighted to find that the store’s number, 233-7226, corresponds to “BEERCAN.” A veterinarian with the phone number 738-2273 would be pleased to know that the number corresponds to the letters “PETCARE.” An automotive dealership would be pleased to know that its phone number, 639-2277, corresponds to “NEWCARS.”

Write a GUI program that, given a seven-digit number, uses a `StreamWriter` object to write to a file every possible seven-letter word combination corresponding to that number. There are 2,187 (3⁷) such combinations. Avoid phone numbers with the digits 0 and 1.

ANS:

```

1 // Ex. 17.7: Phone.cs
2 // Saves all possible word combinations for a
3 // 7-digit phone number to a file
4 using System;
5 using System.IO;
6
7 class Phone
8 {
9     // output letter combinations to file
10    public string Calculate( int phoneNumber, string path )
11    {
12        StreamWriter output = null;
13
14        string[,] letters = { { " ", " ", " " },
15                               { "A", "B", "C" }, { "D", "E", "F" },
16                               { "G", "H", "I" }, { "J", "K", "L" }, { "M", "N", "O" },
17                               { "P", "R", "S" }, { "T", "U", "V" }, { "W", "X", "Y" } };
18
19        int[] digits = new int[ 7 ];
20
21        for ( int i = 6; i >= 0; i-- )
22        {
23            digits[ i ] = ( int )( phoneNumber % 10 );
24            phoneNumber /= 10;
25        } // end for
26
27        if ( path != string.Empty )
28        {
29            output = new StreamWriter( path );
30
31            try
32            {
33                // output all possible combinations
34                for ( int loop1 = 0; loop1 <= 2; loop1++ )
35                {
36                    for ( int loop2 = 0; loop2 <= 2; loop2++ )
37                    {
38                        for ( int loop3 = 0; loop3 <= 2; loop3++ )
39                        {
40                            for ( int loop4 = 0; loop4 <= 2; loop4++ )
41                            {
42                                for ( int loop5 = 0; loop5 <= 2; loop5++ )
43                                {
44                                    for ( int loop6 = 0; loop6 <= 2; loop6++ )
45                                    {
46                                        for ( int loop7 = 0; loop7 <= 2; loop7++ )
47                                        {
48                                            output.WriteLine(
49                                                letters[ digits[ 0 ], loop1 ] +

```

```

50         letters[ digits[ 1 ], loop2 ] +
51         letters[ digits[ 2 ], loop3 ] +
52         letters[ digits[ 3 ], loop4 ] +
53         letters[ digits[ 4 ], loop5 ] +
54         letters[ digits[ 5 ], loop6 ] +
55         letters[ digits[ 6 ], loop7 ] );
56     } // end for
57 } // end for
58 } // end for
59 } // end for
60 } // end for
61 } // end for
62 } // end for
63 } // end try
64 catch ( IOException )
65 {
66     output.Close();
67     return "Error Writing to File.";
68 } // end catch
69
70     output.Close();
71     return "Done";
72 } // end if
73
74     output.Close();
75     return "Invalid path";
76 } // end method Calculate
77 } // end class Phone

```

```

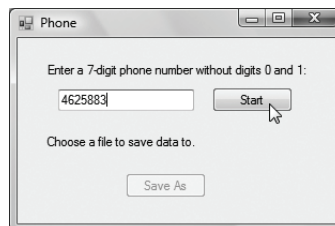
1  // Ex. 17.7: PhoneWordCombinationsForm.cs
2  // Saves all possible word combinations for a
3  // 7-digit phone number to a file
4  using System;
5  using System.Windows.Forms;
6
7  namespace PhoneWordCombinations
8  {
9      public partial class PhoneWordCombinationsForm : Form
10     {
11         string fileName;
12
13         // parameterless constructor
14         public PhoneWordCombinationsForm()
15         {
16             InitializeComponent();
17         } // end constructor
18
19         private void startButton_Click( object sender, EventArgs e )
20         {
21             Phone application = new Phone();
22
23             displayLabel.Text = "Please Wait...";
24             string result = application.Calculate(

```

```

25         Convert.ToInt32( inputTextBox.Text ), fileName );
26         displayLabel.Text = result;
27
28         startButton.Enabled = false;
29         saveButton.Enabled = true;
30     } // end method startButton_Click
31
32     private void saveButton_Click( object sender, EventArgs e )
33     {
34         // create and show dialog box enabling user to save file
35         DialogResult result; // result of SaveFileDialog
36
37         using ( SaveFileDialog fileChooser = new SaveFileDialog() )
38         {
39             // allow user to create file
40             fileChooser.CheckFileExists = false;
41             result = fileChooser.ShowDialog();
42             fileName = fileChooser.FileName; // name of file to save data
43         } // end using
44
45         // ensure that user clicked "OK"
46         if ( result == DialogResult.OK )
47         {
48
49             if ( fileName == string.Empty )
50                 MessageBox.Show( "Invalid File Name",
51                                 "Invalid File name", MessageBoxButtons.OK,
52                                 MessageBoxIcon.Information );
53             else
54                 saveButton.Enabled = false;
55
56             // enable start button
57             startButton.Enabled = true;
58         } // end if
59     } // end method saveButton_Click
60 } // end class PhoneWordCombinationsForm
61 } // end namespace PhoneWordCombinations

```



```
GMAJTDD
GMAJTTE
GMAJTTF
GMAJTUD
GMAJTUE
GMAJTUF
GMAJTV D
GMAJTVE
GMAJTVF
GMAJUTD
GMAJUTE
.
.
.
```

17.8 (Student Poll) Figure 8.8 contains an array of survey responses that's hard-coded into the program. Suppose we wish to process survey results that are stored in a file. First, create a Windows Form that prompts the user for survey responses and outputs each response to a file. Use `StreamWriter` to create a file called `numbers.txt`. Each integer should be written using method `Write`. Then add a `TextBox` that will output the frequency of survey responses. You should modify the code in Fig. 8.8 to read the survey responses from `numbers.txt`. The responses should be read from the file by using a `StreamReader`. Class `string`'s `split` method should be used to split the input string into separate responses, then each response should be converted to an integer. The program should continue to read responses until it reaches the end of file. The results should be output to the `TextBox`.

ANS:

```
1 // Ex. 17.8: StudentPoll.cs
2 // Allow student to take a survey
3 // and view the results in a TextBox
4 using System;
5 using System.Windows.Forms;
6 using System.IO;
7
8 namespace StudentPoll
9 {
10     public partial class StudentPollForm : Form
11     {
12         StreamWriter writer;
13         StreamReader input;
14         int number;
15         bool isFirst;
16
17         // parameterless constructor
18         public StudentPollForm()
19         {
20             InitializeComponent();
21             writer = new StreamWriter( "numbers.txt" );
22             isFirst = true;
23         } // end constructor
24
25         private void inputTextBox_KeyDown( object sender, EventArgs e )
26         {
```

```

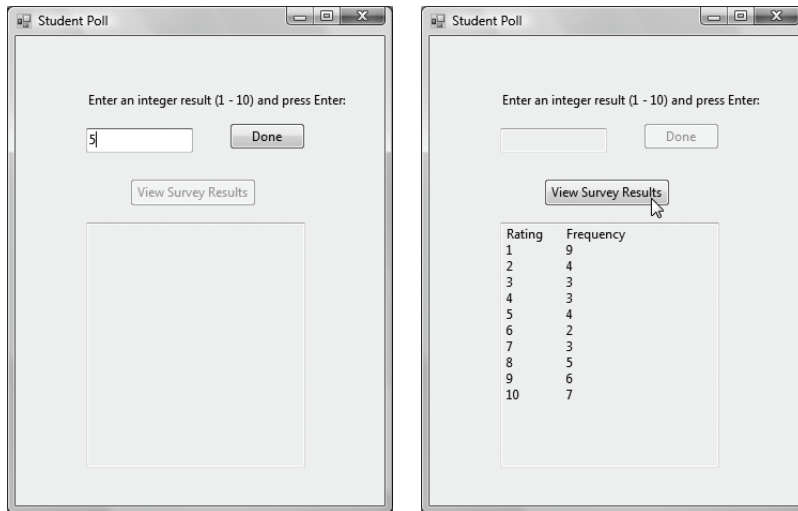
27         if ( e.KeyCode == Keys.Enter )
28         {
29             try
30             {
31                 if ( string.IsNullOrEmpty( inputTextBox.Text ) )
32                 {
33                     MessageBox.Show( "Please fill in the TextBox." );
34                 } // end if
35             } else
36             {
37
38                 number = Convert.ToInt32( inputTextBox.Text );
39
40                 if ( number <= 10 && number >= 1 )
41                 {
42                     if ( isFirst )
43                     {
44                         writer.Write( inputTextBox.Text );
45                         isFirst = false;
46                     } // end if
47                     else
48                     {
49                         writer.Write( "," + inputTextBox.Text );
50                     }
51                 } // end else
52             } // end try
53         } catch ( IOException )
54         {
55             MessageBox.Show( "Error with input." );
56         } // end catch
57
58         inputTextBox.Clear();
59     } // end if
60 }
61
62 private void doneButton_Click( object sender, EventArgs e )
63 {
64     writer.Close();
65     displayButton.Enabled = true;
66     doneButton.Enabled = false;
67     inputTextBox.ReadOnly = true;
68 }
69
70 private void displayButton_Click( object sender, EventArgs e )
71 {
72     input = new StreamReader( "numbers.txt" );
73
74     string inputString = input.ReadToEnd();
75     string[] stringArray;
76     int[] frequency = new int[ 11 ];
77
78     stringArray = inputString.Split( ',' );
79
80     int[] responses = new int[ stringArray.Length ];

```

```

81
82     for ( int index = 0; index < stringArray.Length; index++ )
83         responses[ index ] = Convert.ToInt32( stringArray[ index ] );
84
85     // for each answer, select responses element and use that value
86     // as frequency index to determine element to increment
87     foreach ( int answer in responses )
88         ++frequency[ answer ];
89
90     displayTextBox.Clear();
91     displayTextBox.AppendText( "Rating\tFrequency\n" );
92
93     // output each array element's value
94     for ( int rating = 1; rating < frequency.Length; rating++ )
95         displayTextBox.AppendText( rating + "\t" + frequency[ rating ]
96             + "\n" );
97     } // end method displayButton_Click
98 } // end class StudentPollForm
99 } // end namespace StudentPoll

```



Making a Difference Exercise

17.9 (Phishing Scanner) Phishing is a form of identity theft in which, in an e-mail, a sender posing as a trustworthy source attempts to acquire private information, such as your user names, passwords, credit-card numbers and social security number. Phishing e-mails claiming to be from popular banks, credit-card companies, auction sites, social networks and online payment services may look quite legitimate. These fraudulent messages often provide links to spoofed (fake) websites where you're asked to enter sensitive information.

Visit McAfee® (www.mcafee.com/us/threat_center/anti_phishing/phishing_top10.html), Security Extra (www.securityextra.com/), www.snopes.com and other websites to find lists of the top phishing scams. Also check out the Anti-Phishing Working Group (www.antiphishing.org/),

and the FBI's Cyber Investigations website (www.fbi.gov/cyberinvest/cyberhome.htm), where you'll find information about the latest scams and how to protect yourself.

Create a list of 30 words, phrases and company names commonly found in phishing messages. Assign a point value to each based on your estimate of its likeliness to be in a phishing message (e.g., one point if it's somewhat likely, two points if moderately likely, or three points if highly likely). Write a program that scans a file of text for these terms and phrases. For each occurrence of a keyword or phrase within the text file, add the assigned point value to the total points for that word or phrase. For each keyword or phrase found, output one line with the word or phrase, the number of occurrences and the point total. Then show the point total for the entire message. Does your program assign a high point total to some actual phishing e-mails you've received? Does it assign a high point total to some legitimate e-mails you've received? [Note: If you search online for "sample phishing emails," you'll find many examples of text that you can test with this program.]

ANS: [Note: We placed a file called "phishing.txt" with a sample phishing email in the bin\Debug and bin\Release folders of this solution.]

```

1 // Exercise 17.9 Solution: PhishingScanner.cs
2 // Calculates the phishing score of a message in file phishing.txt
3 using System;
4 using System.IO;
5
6 class PhishingScanner
7 {
8     // list of phishing words
9     static string[] phishingWords = {
10         "amazon", "official", "bank", "security", "urgent", "alert",
11         "important", "information", "ebay", "password", "credit", "verify",
12         "confirm", "account", "bill", "immediately", "address", "telephone",
13         "ssn", "charity", "check", "secure", "personal", "confidential",
14         "atm", "warning", "fraud", "citibank", "irs", "paypal" };
15
16     // parallel array of point values
17     static int[] phishingPoints = { 2, 2, 1, 1, 1, 1, 1, 2, 3,
18         3, 3, 1, 1, 1, 1, 1, 2, 2, 3, 2, 1, 1, 1, 1, 2, 2, 2, 2, 2, 1 };
19
20     static void Main(string[] args)
21     {
22         int[] wordCount = new int[ phishingWords.Length ];
23
24         // count number of times each word occurs in file
25         bool fileReadSuccessfully = countOccurrences( wordCount );
26
27         // display results
28         if ( fileReadSuccessfully )
29             displayResults( wordCount );
30     } // end Main
31
32     static bool countOccurrences( int[] wordCount )
33     {
34         try
35         {
36             // try to open file
37             StreamReader fileReader; // reads data from a text file

```


```

38     FileStream input = new FileStream(
39         "phishing.txt", FileMode.Open, FileAccess.Read );
40     fileReader = new StreamReader( input );
41
42     string email = fileReader.ReadToEnd(); // read contents of file
43
44     // tokenize the email
45     char[] delimiters =
46         { '\n', '\t', ' ', ',', '\r', ':', '!', '.', '?', ';' };
47     string[] words = email.Split( delimiters );
48
49     // read file, one word at a time
50     foreach ( string word in words )
51     {
52         for ( int i = 0; i < phishingWords.Length; i++ )
53             if ( phishingWords[ i ] == word.ToLower() )
54                 ++wordCount[ i ]; // increment occurrence count
55     } // end while
56 } // end try
57 catch ( IOException )
58 {
59     Console.WriteLine( "Error reading from file" );
60     return false;
61 } // end catch
62
63     return true;
64 } // end function countOccurrences
65
66 static void displayResults( int[] wordCount )
67 {
68     int totalPoints = 0;
69
70     // display header
71     Console.WriteLine( "{0,20}{1,10}{2,10}",
72         "Word", "Count", "Points" );
73
74     for ( int i = 0; i < wordCount.Length; i++ )
75     {
76         // don't display if word was not found
77         if ( wordCount[ i ] == 0 )
78             continue;
79
80         int points = wordCount[ i ] * phishingPoints[ i ];
81         totalPoints += points;
82
83         // display count
84         Console.WriteLine( "{0,20}{1,10}{2,10}",
85             phishingWords[ i ], wordCount[ i ], points );
86     } // end for
87
88     Console.WriteLine( "Total points: {0}", totalPoints );
89 } // end function displayResults
90 } // end class PhishingScanner

```

Word	Count	Points
bank	1	1
verify	4	4
account	1	1
address	2	4
atm	1	2
citibank	5	10

Total points: 22



Now go, write it before them in a table, and note it in a book, that it may be for the time to come for ever and ever.

—Isaiah 30:8

It is a capital mistake to theorize before one has data.

—Arthur Conan Doyle

Objectives

In this chapter you'll learn:

- The relational database model.
- To use LINQ to retrieve and manipulate data from a database.
- To add data sources to projects.
- To use the Object Relational Designer to create LINQ to SQL classes.
- To use the IDE's drag-and-drop capabilities to display database tables in applications.
- To use data binding to move data seamlessly between GUI controls and databases.
- To create Master/Detail views that enable you to select a record and display its details.

Self-Review Exercises

18.1 Fill in the blanks in each of the following statements:

- a) A table in a relational database consists of _____ and _____ in which values are stored.

ANS: rows, columns.

- b) The _____ uniquely identifies each row in a relational database table.

ANS: primary key.

- c) A relational database can be manipulated in LINQ to SQL via a(n) _____ object, which contains properties for accessing each table in the database.

ANS: DataContext.

- d) The _____ control (presented in this chapter) displays data in rows and columns that correspond to the rows and columns of a data source.

ANS: DataGridView.

- e) Merging data from multiple relational database tables is called _____ the data.

ANS: joining.

- f) A(n) _____ is a column (or group of columns) in a relational database table that matches the primary-key column (or group of columns) in another table.

ANS: foreign key.

- g) A(n) _____ object serves as an intermediary between a data source and its corresponding data-bound GUI control.

ANS: BindingSource.

- h) The _____ property of a control specifies where it gets the data it displays.

ANS: DataSource.

- i) The _____ clause declares a new temporary variable within a LINQ query.

ANS: Let.

18.2 State whether each of the following is *true* or *false*. If *false*, explain why.

- a) Providing the same value for a foreign key in multiple rows causes the DBMS to report an error.

ANS: False. Multiple rows can have the same value for a foreign key. Providing the same value for the primary key in multiple rows causes the DBMS to report an error, because duplicate primary keys would prevent each row from being identified uniquely.

- b) Providing a foreign-key value that does not appear as a primary-key value in another table is an error.

ANS: True.

- c) The result of a query can be sorted in ascending or descending order.

ANS: True.

- d) A BindingNavigator object can extract data from a database.

ANS: False. A BindingNavigator allows users to browse and manipulate data displayed by another GUI control. A DataContext can extract data from a database.

- e) LINQ to SQL automatically saves changes made back to the database.

ANS: False. You must call the SubmitChanges method of the DataContext to save the changes made back to the database.

Exercises

18.3 (*Display Authors Table Application Modification*) Modify the DisplayTable application in Section 21.5 to contain a TextBox and a Button that allow the user to search for specific authors by last name. Include a Label to identify the TextBox. Using the techniques presented in Section 21.9, create a LINQ query that changes the DataSource property of AuthorBindingSource to contain only the specified authors.

ANS:

```

1  // Ex. 18.3 Solution: DisplayAuthorsTable.cs
2  // Displaying data from a database table in a DataGridView.
3  using System;
4  using System.Linq;
5  using System.Windows.Forms;
6
7  namespace DisplayTable
8  {
9      public partial class DisplayAuthorsTable : Form
10     {
11         // constructor
12         public DisplayAuthorsTable()
13         {
14             DisplayAuthorsTable();
15         } // end constructor
16
17         // LINQ to SQL data context
18         private BooksDataContext database = new BooksDataContext();
19
20         // load data from database into DataGridView
21         private void DisplayAuthorsTable_Load( object sender, EventArgs e )
22         {
23             // use LINQ to order the data for display
24             authorBindingSource.DataSource =
25                 from author in database.Authors
26                 orderby author.AuthorID
27                 select author;
28         } // end method DisplayAuthorsTable_Load
29
30         // Click event handler for the Save Button in the
31         // BindingNavigator saves the changes made to the data
32         private void authorBindingNavigatorSaveItem_Click(
33             object sender, EventArgs e )
34         {
35             Validate(); // validate input fields
36             authorBindingSource.EndEdit(); // indicate edits are complete
37             database.SubmitChanges(); // write changes to database file
38         } // end method authorBindingNavigatorSaveItem_Click
39
40         // displays only rows that have the specified last name
41         private void findButton_Click( object sender, EventArgs e )
42         {
43             // update DataSource to include only people
44             // with specified last name
45             authorBindingSource.DataSource =
46                 from author in database.Authors
47                 where author.LastName.StartsWith( findTextBox.Text )
48                 orderby author.AuthorID
49                 select author;
50         } // end method findButton_Click
51     } // end class DisplayAuthorsTable
52 } // end namespace DisplayTable

```

The 'Display Query Result' window displays a table with the following data:

	ISBN	Title	EditionNumber	Copyright
▶	0136053033	Simply Visual Basic 2008	3	2009
	013605305X	Visual Basic 2008 How to Progr...	4	2009
	013605322X	Visual C# 2008 How to Program	3	2009
	0136151574	Visual C++ 2008 How to Program	2	2008
*				

Below the table is a search section with a dropdown menu set to 'All titles', a 'Title search' label, a 'Search Term' text box containing 'Visual', and a 'Find' button.

The 'Display Table' window displays a table with the following data:

	AuthorID	FirstName	LastName
▶	3	Greg	Ayer
*			

Below the table is a search section with a label 'Find authors by last name', a 'Last name' text box containing 'Ayer', and a 'Find' button.

18.4 (*Display Query Results Application Modification*) Modify the **Display Query Results** application in Section 21.6 to contain a `TextBox` and a `Button` that allow the user to perform a search of the book titles in the `Titles` table of the `Books` database. Use a `Label` to identify the `TextBox`. When the user clicks the `Button`, the application should execute and display the result of a query that selects all the rows in which the search term entered by the user in the `TextBox` appears anywhere in the `Title` column. For example, if the user enters the search term “Visual,” the `DataGridView` should display the rows for *Simply Visual Basic 2008*, *Visual Basic 2008 How to Program*, *Visual C# 2008 How to Program* and *Visual C++ 2008 How to Program*. If the user enters “Simply,” the `DataGridView` should display only the row for *Simply Visual Basic 2008*. [Hint: Use the `Contains` method of the `String` class.]

ANS:

```

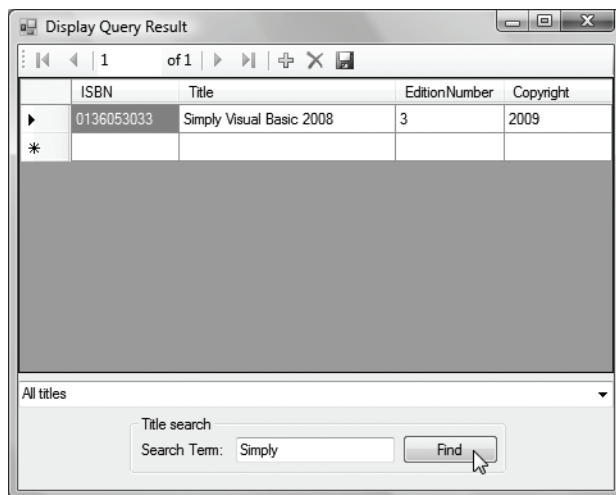
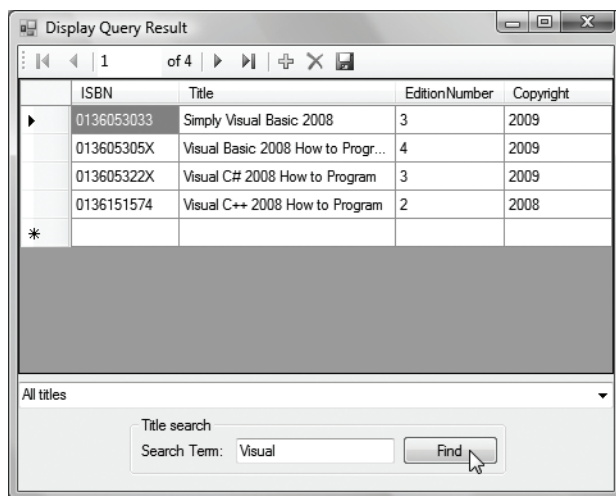
1  // Ex. 18.4 Solution: TitleQueries.cs
2  // Displaying the result of a user-selected query in a DataGridView.
3  using System;
4  using System.Linq;
5  using System.Windows.Forms;
6
7  namespace DisplayQueryResult
8  {
9      public partial class TitleQueries : Form
10     {
11         // constructor
12         public TitleQueries()
13         {
14             InitializeComponent();
15         } // end constructor
16
17         // LINQ to SQL data context
18         private BooksDataContext database = new BooksDataContext();
19
20         // load data from database into DataGridView
21         private void TitleQueries_Load(
22             object sender, EventArgs e )
23         {
24             // write SQL to standard output stream
25             database.Log = Console.Out;
26
27             // set the ComboBox to show the default query that
28             // selects all books from the Titles table
29             queriesComboBox.SelectedIndex = 0;
30         } // end class TitleQueries_Load
31
32         // Click event handler for the Save Button in the
33         // BindingNavigator saves the changes made to the data
34         private void titleBindingNavigatorSaveItem_Click(
35             object sender, EventArgs e )
36         {
37             Validate(); // validate input fields
38             titleBindingSource.EndEdit(); // indicate edits are complete
39             database.SubmitChanges(); // write changes to database file
40
41             // when saving, return to "all titles" query
42             queriesComboBox.SelectedIndex = 0;
43         } // end method titleBindingNavigatorSaveItem_Click
44
45         // loads data into titleBindingSource based on user-selected query
46         private void queriesComboBox_SelectedIndexChanged(
47             object sender, EventArgs e )
48         {
49             // set the data displayed according to what is selected
50             switch ( queriesComboBox.SelectedIndex )
51             {

```

```

52         case 0: // all titles
53             // use LINQ to order the books by title
54             titleBindingSource.DataSource =
55                 from title in database.Titles
56                 orderby title.Title1
57                 select title;
58             break;
59         case 1: // titles with 2008 copyright
60             // use LINQ to get titles with 2008
61             // copyright and sort them by title
62             titleBindingSource.DataSource =
63                 from title in database.Titles
64                 where title.Copyright == "2008"
65                 orderby title.Title1
66                 select title;
67             break;
68         case 2: // titles ending with "How to Program"
69             // use LINQ to get titles ending with
70             // "How to Program" and sort them by title
71             titleBindingSource.DataSource =
72                 from title in database.Titles
73                 where title.Title1.EndsWith( "How to Program" )
74                 orderby title.Title1
75                 select title;
76             break;
77     } // end switch
78
79     titleBindingSource.MoveFirst(); // move to first entry
80 } // end method queriesComboBox_SelectedIndexChanged
81
82 // display only titles that contain the search term
83 private void findButton_Click( object sender, EventArgs e )
84 {
85     // filter titles based on search term in findTextBox
86     titleBindingSource.DataSource =
87         from title in database.Titles
88         where title.Title1.Contains( findTextBox.Text )
89         orderby title.Title1
90         select title;
91     } // end method findButton_Click
92 } // end class TitleQueries
93 } // end namespace DisplayQueryResult

```



18.5 (Baseball Database Application) Build an application that executes a query against the Players table of the Baseball database included in the Databases folder with this chapter's examples. Display the table in a DataGridView, and add a TextBox and Button to allow the user to search for a specific player by last name. Use a Label to identify the TextBox. Clicking the Button should execute the appropriate query.

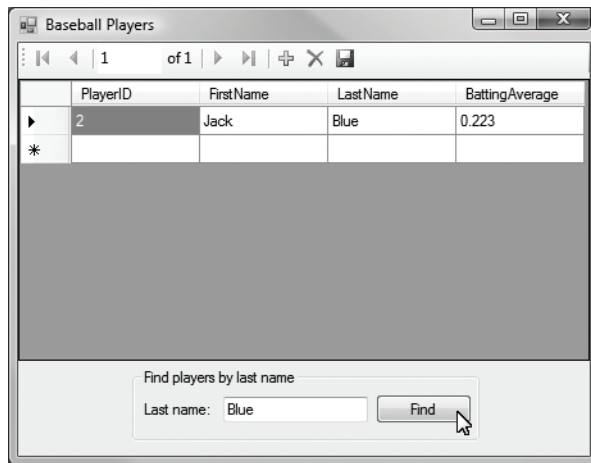
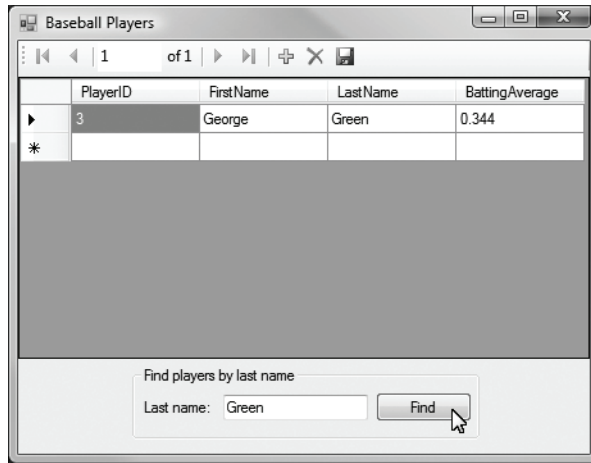
ANS:

```

1 // Ex. 18.5 Solution: BaseballPlayersForm.cs
2 // Displays the Players table of the Baseball database in a DataGridView
3 // and allows the user to search for players by last name.
4 using System;

```

```
5 using System.Linq;
6 using System.Windows.Forms;
7
8 namespace BaseballPlayers
9 {
10     public partial class BaseballPlayersForm : Form
11     {
12         // constructor
13         public BaseballPlayersForm()
14         {
15             InitializeComponent();
16         } // end constructor
17
18         // database connection
19         private BaseballDataContext database = new BaseballDataContext();
20
21         // load the data from the database when the Form loads
22         private void BaseballPlayersForm_Load( object sender, EventArgs e )
23         {
24             // fill the DataGridView with the player information
25             playerBindingSource.DataSource =
26                 from player in database.Players
27                 orderby player.PlayerID
28                 select player;
29         } // end method BaseballPlayersForm_Load
30
31         // Click event handler for the Save Button in the
32         // BindingNavigator saves the changes made to the data
33         private void playerBindingNavigatorSaveItem_Click(
34             object sender, EventArgs e )
35         {
36             Validate(); // validate input fields
37             playerBindingSource.EndEdit(); // indicate edits are complete
38             database.SubmitChanges(); // write changes to database file
39         } // end method playerBindingNavigatorSaveItem_Click
40
41         // filter to display only players with the selected last name
42         private void findButton_Click( object sender, EventArgs e )
43         {
44             // include only players that have the last name in findTextBox
45             playerBindingSource.DataSource =
46                 from player in database.Players
47                 where player.LastName.StartsWith( findTextBox.Text )
48                 orderby player.PlayerID
49                 select player;
50         } // end method findButton_Click
51     } // end class BaseballPlayersForm
52 } // end namespace BaseballPlayers
```



18.6 (Baseball Database Application Modification) Modify Exercise 18.5 to allow the user to locate players with batting averages in a specific range. Add a `minimumTextBox` for the minimum batting average (0.000 by default) and a `maximumTextBox` for the maximum batting average (1.000 by default). Use a `Label` to identify each `TextBox`. Add a `Button` for executing a query that selects rows from the `Players` table in which the `BattingAverage` column is greater than or equal to the specified minimum value and less than or equal to the specified maximum value.

ANS:

```
1 // Ex. 18.6 Solution: BaseballPlayersForm.cs
2 // Displays the Players table of the Baseball database in a DataGridView
3 // and allows the user to search for players by name or batting average.
4 using System;
5 using System.Linq;
```

```

6  using System.Windows.Forms;
7
8  namespace BaseballPlayers
9  {
10     public partial class BaseballPlayersForm : Form
11     {
12         // constructor
13         public BaseballPlayersForm()
14         {
15             InitializeComponent();
16         } // end constructor
17
18         // database connection
19         private BaseballDataContext database = new BaseballDataContext();
20
21         // load the data from the database when the Form loads
22         private void BaseballPlayersForm_Load( object sender, EventArgs e )
23         {
24             // fill the DataGridView with the player information
25             playerBindingSource.DataSource =
26                 from player in database.Players
27                 orderby player.PlayerID
28                 select player;
29         } // end method BaseballPlayersForm_Load
30
31         // Click event handler for the Save Button in the
32         // BindingNavigator saves the changes made to the data
33         private void playerBindingNavigatorSaveItem_Click(
34             object sender, EventArgs e )
35         {
36             Validate(); // validate input fields
37             playerBindingSource.EndEdit(); // indicate edits are complete
38             database.SubmitChanges(); // write changes to database file
39         } // end method playerBindingNavigatorSaveItem_Click
40
41         // filter to display only players with the selected last name
42         private void findButton_Click( object sender, EventArgs e )
43         {
44             // include only players that have the last name in findTextBox
45             playerBindingSource.DataSource =
46                 from player in database.Players
47                 where player.LastName.StartsWith( findTextBox.Text )
48                 orderby player.PlayerID
49                 select player;
50         } // end method findButton_Click
51
52         // filter to display only players with
53         // batting averages in the specified range
54         private void averageButton_Click( object sender, EventArgs e )
55         {
56             // convert strings in TextBoxes to decimals
57             decimal minimum = Convert.ToDecimal( minimumTextBox.Text );
58             decimal maximum = Convert.ToDecimal( maximumTextBox.Text );
59

```

```

60         // use LINQ to select only players in the range given in
61         // minimumTextBox and maximumTextBox
62         playerBindingSource.DataSource =
63             from player in database.Players
64             where player.BattingAverage >= minimum
65                   && player.BattingAverage <= maximum
66             orderby player.PlayerID
67             select player;
68     } // end method averageButton_Click
69 } // end class BaseballPlayersForm
70 } // end namespace BaseballPlayers

```

The screenshot shows the 'Baseball Players' application window. At the top, it says '1 of 1'. Below this is a table with the following data:

PlayerID	FirstName	LastName	BattingAverage
2	Jack	Blue	0.223

Below the table is a search section with two options:

- Find players by last name:** A text box labeled 'Last name:' and a 'Find' button.
- Find players by batting average:** Two text boxes labeled 'Minimum:' (containing '0.0') and 'Maximum:' (containing '0.3'), followed by a 'Find' button.

The screenshot shows the 'Baseball Players' application window. At the top, it says '1 of 2'. Below this is a table with the following data:

PlayerID	FirstName	LastName	BattingAverage
1	John	Red	0.375
3	George	Green	0.344

Below the table is a search section with two options:

- Find players by last name:** A text box labeled 'Last name:' and a 'Find' button.
- Find players by batting average:** Two text boxes labeled 'Minimum:' (containing '0.3') and 'Maximum:' (containing '0.5'), followed by a 'Find' button.

18.7 (Project: AdventureWorks Sample Database) In this exercise, use Microsoft's sample AdventureWorks database. There are several versions available, depending on what version of SQL Server you're using and your operating system. We used the AdventureWorks LT version of the database—a smaller version with fewer tables and less data than the full version. The files for SQL Server 2008 can be downloaded from

msftdbprodsamples.codeplex.com/Release/ProjectReleases.aspx?ReleaseId=24854

The installer allows you to select which version of the database to install.

Use the AdventureWorks database in an application that runs multiple queries on the database and displays the results. First, it should list customers and their addresses. As this is a large list, limit the number of results to ten. [*Hint: Use LINQ's Take clause at the end of the query to return a limited number of results. The Take clause consists of the Take operator, then an Integer specifying how many rows to take.*] Second, if a category has subcategories, the output should show the category with its subcategories indented below it. The queries described here require the AdventureWorks tables Address, Customer, CustomerAddress and ProductCategory.

ANS:

```

1  // Ex. 18.7 Solution: AdventureWorks.cs
2  // Using LINQ to aggregate data from multiple
3  // tables in the AdventureWorks database.
4  using System;
5  using System.Linq;
6
7  namespace AdventureWorks
8  {
9      public class AdventureWorks
10     {
11         public static void Main( string[] args )
12         {
13             // create the database connection
14             AdventureWorksDataContext database =
15                 new AdventureWorksDataContext();
16
17             // use LINQ to retrieve first 10 customer-address pairs
18             var customersAndAddresses =
19                 from customerAndAddress in database.CustomerAddresses
20                 let customer = customerAndAddress.Customer
21                 let address = customerAndAddress.Address
22                 select new
23                 {
24                     Name = customer.FirstName + " " + customer.LastName,
25                     Address = string.Format( "{0}, {1}, {2} {3}",
26                                             address.AddressLine1, address.City,
27                                             address.StateProvince, address.PostalCode )
28                 };
29
30             // display customers and addresses
31             foreach ( var element in customersAndAddresses.Take( 10 ) )
32             {
33                 Console.WriteLine( "{0,-20} {1}",
34                                     element.Name, element.Address );
35             } // end foreach
36
37             Console.WriteLine(); // blank line for readability
38

```

```

39         // use LINQ to retrieve categories with subcategories
40         // include only categories with subcategories
41         var categories =
42             from category in database.ProductCategories
43             let children = category.ProductCategories
44             where children.Any()
45             select new
46             {
47                 category.Name,
48                 Children = ( from child in children select child.Name )
49             };
50
51         // display categories and their subcategories
52         foreach ( var category in categories )
53         {
54             // display category name
55             Console.WriteLine( category.Name + ":" );
56
57             foreach ( var child in category.Children )
58             {
59                 // display subcategory name
60                 Console.WriteLine( "\t" + child );
61             } // end inner foreach
62         } // end outer foreach
63     } // end Main
64 } // end class AdventureWorks
65 } // end namespace AdventureWorks

```

Orlando Gee	2251 Elliot Avenue, Seattle, Washington 98104
Keith Harris	7943 Walnut Ave, Renton, Washington 98055
Keith Harris	3207 S Grady Way, Renton, Washington 98055
Donna Carreras	12345 Sterling Avenue, Irving, Texas 75061
Janet Gates	800 Interchange Blvd., Austin, Texas 78701
Janet Gates	165 North Main, Austin, Texas 78701
Lucy Harrington	482505 Warm Springs Blvd., Fremont, California 94536
Rosmarie Carroll	39933 Mission Oaks Blvd, Camarillo, California 93010
Dominic Gash	5420 West 22500 South, Salt Lake City, Utah 84101
Kathleen Garza	6388 Lake City Way, Burnaby, British Columbia V5A 3A6

Bikes:

- Mountain Bikes
- Road Bikes
- Touring Bikes

Components:

- Handlebars
- Bottom Brackets
- Brakes
- Chains
- Cranksets
- Derailleurs
- Forks
- Headsets
- Mountain Frames
- Pedals
- Road Frames

```

        Saddles
        Touring Frames
        Wheels
Clothing:
        Bib-Shorts
        Caps
        Gloves
        Jerseys
        Shorts
        Socks
        Tights
        Vests
Accessories:
        Bike Racks
        Bike Stands
        Bottles and Cages
        Cleaners
        Fenders
        Helmets
        Hydration Packs
        Lights
        Locks
        Panniers
        Pumps
        Tires and Tubes

```

18.8 (*Project: AdventureWorks Master/Detail view*) Use the Microsoft AdventureWorks database from Exercise 18.7 to create a master/detail view. One master list should be customers, and the other should be products—these should show the details of products the customers purchased, and customers who purchased those products, respectively. Note that there are many customers in the database who did not order any products, and many products that no one ordered. Restrict the drop-down lists so that only customers that have submitted at least one order and products that have been included in at least one order are displayed. The queries in this exercise require the Customer, Product, SalesOrderHeader and SalesOrderDetail tables.

ANS:

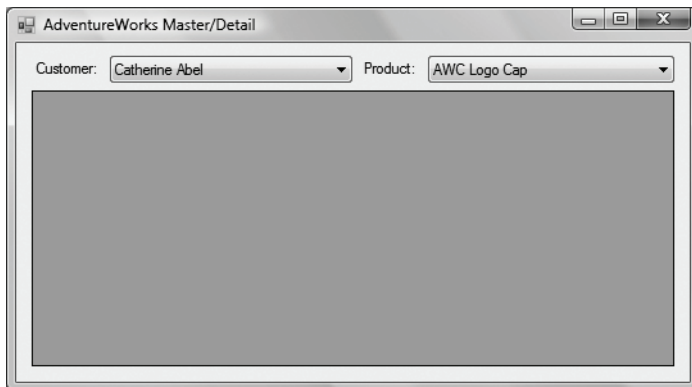
```

1  // Ex. 18.8 Solution: AdventureWorksMasterDetailForm.cs
2  // A master/detail view using the AdventureWorks database.
3  using System;
4  using System.Linq;
5  using System.Windows.Forms;
6
7  namespace AdventureWorksMasterDetail
8  {
9      public partial class AdventureWorksMasterDetailForm : Form
10     {
11         // constructor
12         public AdventureWorksMasterDetailForm()
13         {
14             InitializeComponent();
15         } // end constructor
16

```

```
17 // connection to the database
18 private AdventureWorksDataContext database =
19     new AdventureWorksDataContext();
20
21 // this class allows us to show the
22 // customer's full name in the ComboBox
23 private class CustomerBinding
24 {
25     public Customer Customer { get; set; } // contained customer
26     public string Name { get; set; } // customer's full name
27 } // end class CustomerBinding
28
29 // initialize data sources when the form is loaded
30 private void AdventureWorksMasterDetailForm_Load(
31     object sender, EventArgs e )
32 {
33     customerComboBox.DisplayMember = "Name"; // display full name
34
35     // set the customerComboBox's DataSource
36     // to the list of customers
37     customerComboBox.DataSource =
38         from customer in database.Customers
39         where customer.SalesOrderHeaders.Any()
40         orderby customer.LastName, customer.FirstName
41         let name = customer.FirstName + " " + customer.LastName
42         select
43             new CustomerBinding { Customer = customer, Name = name };
44
45     // display product's name
46     productComboBox.DisplayMember = "Name";
47
48     // set the productComboBox's DataSource to the list of products
49     productComboBox.DataSource =
50         from product in database.Products
51         where product.SalesOrderDetails.Any()
52         orderby product.Name
53         select product;
54
55     // initially, display no "detail" data
56     adventureWorksBindingSource.DataSource = null;
57
58     // set the DataSource of the DataGridView to the BindingSource
59     adventureWorksDataGridView.DataSource =
60         adventureWorksBindingSource;
61 } // end method AdventureWorksMasterDetailForm_Load
62
63 // display products that the customer purchased
64 private void customerComboBox_SelectedIndexChanged(
65     object sender, EventArgs e )
66 {
67     // get the selected Customer object from the ComboBox
68     Customer currentCustomer =
69         ( ( CustomerBinding ) customerComboBox.SelectedItem )
70         .Customer;
```

```
71
72     // traverse the order tables to get
73     // products the customer ordered
74     adventureWorksBindingSource.DataSource =
75         from header in currentCustomer.SalesOrderHeaders
76         from detail in header.SalesOrderDetails
77         select detail.Product;
78 } // end method customerComboBox_SelectedIndexChanged
79
80 // display customers that purchased a specific product
81 private void productComboBox_SelectedIndexChanged(
82     object sender, EventArgs e )
83 {
84     // get the selected Product object from the ComboBox
85     Product currentProduct =
86         ( Product ) productComboBox.SelectedItem;
87
88     // traverse order tables to get customers
89     // that ordered this product
90     adventureWorksBindingSource.DataSource =
91         from order in currentProduct.SalesOrderDetails
92         select order.SalesOrderHeader.Customer;
93 } // end method productComboBox_SelectedIndexChanged
94 } // end class AdventureWorksMasterDetailForm
95 } // end namespace AdventureWorksMasterDetail
```



AdventureWorks Master/Detail

Customer: Richard Byham Product: AWC Logo Cap

	ProductID	Name	ProductNumber	Color	StandardC
▶	876	Hitch Rack - 4-Bi...	RA-H123		44.8800
	884	Short-Sleeve Cla...	SJ-0194-X	Yellow	41.5723
	864	Classic Vest, S	VE-C304-S	Blue	23.7490


AdventureWorks Master/Detail

Customer: Richard Byham Product: Front Brakes

	CustomerID	NameStyle	Title	FirstName	MiddleNam
▶	582	<input type="checkbox"/>	Ms.	Catherine	R.
	295	<input type="checkbox"/>	Mr.	Joseph	P.
	187	<input type="checkbox"/>	Mr.	Frank	
	186	<input type="checkbox"/>	Mr.	Pei	
	223	<input type="checkbox"/>	Ms.	Joyce	
	52	<input type="checkbox"/>	Ms.	Rebecca	
	502	<input type="checkbox"/>	Mr.	Krishna	

Web App Development with ASP.NET: Solutions

19



... the challenges are for the designers of these applications: to forget what we think we know about the limitations of the Web, and begin to imagine a wider, richer range of possibilities. It's going to be fun.

—Jesse James Garrett

If any man will draw up his case, and put his name at the foot of the first page, I will give him an immediate reply. Where he compels me to turn over the sheet, he must wait my leisure.

—Lord Sandwich

Objectives

In this chapter you'll learn:

- Web application development using ASP.NET.
- To handle the events from a Web Form's controls.
- To use validation controls to ensure that data is in the correct format before it's sent from a client to the server.
- To maintain user-specific information.
- To create a data-driven web application using ASP.NET and LINQ to SQL.

Self-Review Exercises


- 19.1** State whether each of the following is *true* or *false*. If *false*, explain why.
- a) Web Form file names end in .aspx.
ANS: True.
 - b) App.config is a file that stores configuration settings for an ASP.NET web application.
ANS: False. Web.config is the file that stores configuration settings for an ASP.NET web application.
 - c) A maximum of one validation control can be placed on a Web Form.
ANS: False. An unlimited number of validation controls can be placed on a Web Form.
 - d) A LinqDataSource control allows a web application to interact with a database.
ANS: True.
- 19.2** Fill in the blanks in each of the following statements:
- a) Web applications contain three basic tiers: _____, _____, and _____.
ANS: bottom (information), middle (business logic), top (client).
 - b) The _____ web control is similar to the ComboBox Windows control.
ANS: DropDownList.
 - c) A control which ensures that the data in another control is in the correct format is called a(n) _____.
ANS: validator.
 - d) A(n) _____ occurs when a page requests itself.
ANS: postback.
 - e) Every ASP.NET page inherits from class _____.
ANS: Page.
 - f) The _____ file contains the functionality for an ASP.NET page.
ANS: code-behind.

Exercises

NOTE: Solutions to the programming exercises are located in the sol_ch19 folder. No Solutions are provided for the Projects in Exercises 19.6 and 19.7.

Searching and Sorting: Solutions

20



*With sobs and tears
he sorted out
Those of the largest size ...*
—Lewis Carroll

*Attempt the end, and never
stand to doubt;
Nothing's so hard, but search
will find it out.*

—Robert Herrick

*It is an immutable law in
business that words are words,
explanations are explanations,
promises are promises — but
only performance is reality.*

—Harold S. Green

Objectives

In this chapter you'll learn:

- To search for a given value in an array using the linear search and binary search algorithm.
- To sort arrays using the iterative selection and insertion sort algorithms.
- To sort arrays using the recursive merge sort algorithm.
- To determine the efficiency of searching and sorting algorithms.

Self-Review Exercises

20.1 Fill in the blanks in each of the following statements:

- a) A selection sort application would take approximately _____ times as long to run on a 128-element array as on a 32-element array.

ANS: 16, because an $O(n^2)$ algorithm takes 16 times as long to sort four times as much information.

- b) The efficiency of merge sort is _____.

ANS: $O(n \log n)$.

20.2 What key aspect of both the binary search and the merge sort accounts for the logarithmic portion of their respective Big Os?

ANS: Both of these algorithms incorporate “halving”—somehow reducing something by half. The binary search eliminates from consideration one half of the array after each comparison. The merge sort splits the array in half each time it is called.

20.3 In what sense is the insertion sort superior to the merge sort? In what sense is the merge sort superior to the insertion sort?

ANS: The insertion sort is easier to understand and to program than the merge sort. The merge sort is far more efficient ($O(n \log n)$) than the insertion sort ($O(n^2)$).

20.4 In the text, we say that after the merge sort splits the array into two subarrays, it then sorts these two subarrays and merges them. Why might someone be puzzled by our statement that “it then sorts these two subarrays”?

ANS: In a sense, it does not really sort these two subarrays. It simply keeps splitting the original array in half until it provides a one-element subarray, which is, of course, sorted. It then builds up the original two subarrays by merging these one-element arrays to form larger subarrays, which are then merged until the whole array has been sorted.

Exercises

20.5 (*Bubble Sort*) Implement the bubble sort—another simple, yet inefficient, sorting technique. It is called bubble sort or sinking sort because smaller values gradually “bubble” their way to the top of the array (i.e., toward the first element) like air bubbles rising in water, while the larger values sink to the bottom (end) of the array. The technique uses nested loops to make several passes through the array. Each pass compares successive pairs of elements. If a pair is in increasing order (or the values are equal), the bubble sort leaves the values as they are. If a pair is in decreasing order, the bubble sort swaps their values in the array.

The first pass compares the first two elements of the array and swaps them if necessary. It then compares the second and third elements. The end of this pass compares the last two elements in the array and swaps them if necessary. After one pass, the largest element will be in the last position. After two passes, the largest two elements will be in the last two positions. Explain why bubble sort is an $O(n^2)$ algorithm.

ANS: Bubble sort contains two nested for loops. The outer for loop (lines 24–33) iterates over `data.Length - 1` passes. The inner for loop (lines 27–32) iterates over `data.Length - 1` elements in the array. When loops are nested, the respective orders are multiplied. This is because for each of $O(n)$ iterations of the outside loop, there are $O(n)$ iterations of the inner loop. This results in a total Big O of $O(n^2)$.

```
1 // Exercise 25.5 Solution: BubbleSort.cs
2 // Sort an array's values into ascending order.
3 using System;
```

```
4
5 public class BubbleSort
6 {
7     private int[] data; // array of values
8     private static Random generator = new Random();
9
10    // create array of given size and fill with random integers
11    public BubbleSort( int size )
12    {
13        data = new int[ size ]; // create space for array
14
15        // fill array with random ints in range 10-99
16        for ( int i = 0; i < size; i++ )
17            data[ i ] = generator.Next( 10, 100 );
18    } // end BubbleSort constructor
19
20    // sort elements of array with bubble sort
21    public void Sort()
22    {
23        // loop for data.Length - 1 passes
24        for ( int pass = 1; pass < data.Length; pass++ )
25        {
26            // loop over elements in array
27            for ( int index = 0; index < data.Length - 1; index++ )
28            {
29                // swap adjacent elements if first is greater than second
30                if ( data[ index ] > data[ index + 1 ] )
31                    Swap( index, index + 1 ); // swap adjacent elements
32            } // end inner for
33        } // end outer for
34    } // end method Sort
35
36    // helper method to swap values in two elements
37    public void Swap( int first, int second )
38    {
39        int temporary = data[ first ]; // store first in temporary
40        data[ first ] = data[ second ]; // replace first with second
41        data[ second ] = temporary; // put temporary in second
42    } // end method Swap
43
44    // method to output values in array
45    public override string ToString()
46    {
47        string temporary = string.Empty;
48
49        // iterate through array
50        foreach ( var element in data )
51            temporary += element + " ";
52
53        temporary += "\n"; // add newline character
54        return temporary;
55    } // end method ToString
56 } // end class BubbleSort
```

```

1 // Exercise 25.5 Solution: BubbleSortTest.cs
2 // Test the bubble sort class.
3 using System;
4
5 public class BubbleSortTest
6 {
7     public static void Main( string[] args )
8     {
9         // create object to perform bubble sort
10        BubbleSort sortArray = new BubbleSort( 10 );
11
12        Console.WriteLine( "Before:" );
13        Console.WriteLine( sortArray ); // display unsorted array
14
15        sortArray.Sort(); // sort array
16
17        Console.WriteLine( "After:" );
18        Console.WriteLine( sortArray ); // display sorted array
19    } // end Main
20 } // end class BubbleSortTest

```

```

Before:
86 55 67 76 61 34 83 27 73 77

After:
27 34 55 61 67 73 76 77 83 86

```

20.6 (*Enhanced Bubble Sort*) Make the following simple modifications to improve the performance of the bubble sort you developed in Exercise 25.5:

- a) After the first pass, the largest number is guaranteed to be in the highest-numbered element of the array; after the second pass, the two highest numbers are “in place”; and so on. Instead of making nine comparisons on every pass, modify the bubble sort to make eight comparisons on the second pass, seven on the third pass and so on.
- b) The data in the array may already be in the proper order or near-proper order, so why make nine passes if fewer will suffice? Modify the sort to check at the end of each pass whether any swaps have been made. If none have been made, the data must already be in the proper order, so the application should terminate. If swaps have been made, at least one more pass is needed.

ANS:

```

1 // Exercise 25.6a Solution: BubbleSort.cs
2 // Sort an array's values into ascending order.
3 using System;
4
5 public class BubbleSort
6 {
7     private int[] data; // array of values
8     private static Random generator = new Random();
9
10    // create array of given size and fill with random integers
11    public BubbleSort( int size )
12    {

```

```

13     data = new int[ size ]; // create space for array
14
15     // fill array with random ints in range 10-99
16     for ( int i = 0; i < size; i++ )
17         data[ i ] = generator.Next( 10, 100 );
18 } // end BubbleSort constructor
19
20 // sort elements of array with bubble sort
21 public void Sort()
22 {
23     // loop for data.Length - 1 passes
24     for ( int pass = 1; pass < data.Length; pass++ )
25     {
26         // loop over elements in array
27         for ( int index = 0; index < data.Length - pass; index++ )
28         {
29             // swap adjacent elements if first is greater than second
30             if ( data[ index ] > data[ index + 1 ] )
31                 Swap( index, index + 1 ); // swap adjacent elements
32         } // end inner for
33     } // end outer for
34 } // end method Sort
35
36 // helper method to swap values in two elements
37 public void Swap( int first, int second )
38 {
39     int temporary = data[ first ]; // store first in temporary
40     data[ first ] = data[ second ]; // replace first with second
41     data[ second ] = temporary; // put temporary in second
42 } // end method Swap
43
44 // method to output values in array
45 public override string ToString()
46 {
47     string temporary = string.Empty;
48
49     // iterate through array
50     foreach ( var element in data )
51         temporary += element + " ";
52
53     temporary += "\n"; // add newline character
54     return temporary;
55 } // end method ToString
56 } // end class BubbleSort

```

```

1 // Exercise 25.6a Solution: BubbleSortTest.cs
2 // Test the bubble sort class.
3 using System;
4
5 public class BubbleSortTest
6 {
7     public static void Main( string[] args )
8     {

```

```

9      // create object to perform bubble sort
10     BubbleSort sortArray = new BubbleSort( 10 );
11
12     Console.WriteLine( "Before:" );
13     Console.WriteLine( sortArray ); // display unsorted array
14
15     sortArray.Sort(); // sort array
16
17     Console.WriteLine( "After:" );
18     Console.WriteLine( sortArray ); // display sorted array
19 } // end Main
20 } // end class BubbleSortTest

```

<p>Before: 13 60 10 94 48 78 41 84 36 23</p> <p>After: 10 13 23 36 41 48 60 78 84 94</p>
--

```

1  // Exercise 25.6b Solution: BubbleSort.cs
2  // Sort an array's values into ascending order.
3  using System;
4
5  public class BubbleSort
6  {
7      private int[] data; // array of values
8      private static Random generator = new Random();
9
10     // create array of given size and fill with random integers
11     public BubbleSort( int size )
12     {
13         data = new int[ size ]; // create space for array
14
15         // fill array with random ints in range 10-99
16         for ( int i = 0; i < size; i++ )
17             data[ i ] = generator.Next( 10, 100 );
18     } // end BubbleSort constructor
19
20     // sort elements of array with bubble sort
21     public void Sort()
22     {
23         // did a swap take place during pass
24         bool didSwap;
25
26         // loop for data.Length - 1 passes
27         for ( int pass = 1; pass < data.Length; pass++ )
28         {
29             didSwap = false; // no swaps have been done for this pass
30
31             // loop over elements in array
32             for ( int index = 0; index < data.Length - pass; index++ )
33             {

```

```

34         // swap adjacent elements if first is greater than second
35         if ( data[ index ] > data[ index + 1 ] )
36         {
37             Swap( index, index + 1 ); // swap adjacent elements
38             didSwap = true; // indicate swap happened
39         } // end if
40     } // end inner for
41
42     // if no swaps, terminate bubble sort
43     if ( !didSwap )
44         return;
45     } // end outer for
46 } // end method Sort
47
48 // helper method to swap values in two elements
49 public void Swap( int first, int second )
50 {
51     int temporary = data[ first ]; // store first in temporary
52     data[ first ] = data[ second ]; // replace first with second
53     data[ second ] = temporary; // put temporary in second
54 } // end method Swap
55
56 // method to output values in array
57 public override string ToString()
58 {
59     string temporary = string.Empty;
60
61     // iterate through array
62     foreach ( var element in data )
63         temporary += element + " ";
64
65     temporary += "\n"; // add newline character
66     return temporary;
67 } // end method ToString
68 } // end class BubbleSort

```

```

1 // Exercise 25.6b Solution: BubbleSortTest.cs
2 // Test the bubble sort class.
3 using System;
4
5 public class BubbleSortTest
6 {
7     public static void Main( string[] args )
8     {
9         // create object to perform bubble sort
10        BubbleSort sortArray = new BubbleSort( 10 );
11
12        Console.WriteLine( "Before:" );
13        Console.WriteLine( sortArray ); // display unsorted array
14
15        sortArray.Sort(); // sort array
16
17        Console.WriteLine( "After:" );

```

```

18      Console.WriteLine( sortArray ); // display sorted array
19  } // end Main
20 } // end class BubbleSortTest

```

<p>Before: 89 26 20 22 50 54 99 33 52 32</p> <p>After: 20 22 26 32 33 50 52 54 89 99</p>
--

20.7 (*Bucket Sort*) A bucket sort begins with a one-dimensional array of positive integers to be sorted and a two-dimensional array of integers with rows indexed from 0 to 9 and columns indexed from 0 to $n - 1$, where n is the number of values to be sorted. Each row of the two-dimensional array is referred to as a *bucket*. Write a class named `BucketSort` containing a method called `Sort` that operates as follows:

- Place each value of the one-dimensional array into a row of the bucket array, based on the value's "ones" (rightmost) digit. For example, 97 is placed in row 7, 3 is placed in row 3 and 100 is placed in row 0. This procedure is called a *distribution pass*.
- Loop through the bucket array row by row, and copy the values back to the original array. This procedure is called a *gathering pass*. The new order of the preceding values in the one-dimensional array is 100, 3 and 97.
- Repeat this process for each subsequent digit position (tens, hundreds, thousands, and so on).

On the second (tens digit) pass, 100 is placed in row 0, 3 is placed in row 0 (because 3 has no tens digit) and 97 is placed in row 9. After the gathering pass, the order of the values in the one-dimensional array is 100, 3 and 97. On the third (hundreds digit) pass, 100 is placed in row 1, 3 is placed in row 0 and 97 is placed in row 0 (after the 3). After this last gathering pass, the original array is in sorted order.

Note that the two-dimensional array of buckets is 10 times the length of the integer array being sorted. This sorting technique provides better performance than a bubble sort, but requires much more memory—the bubble sort requires space for only one additional element of data. This comparison is an example of the space/time trade-off: The bucket sort uses more memory than the bubble sort, but performs better. This version of the bucket sort requires copying all the data back to the original array on each pass. Another possibility is to create a second two-dimensional bucket array and repeatedly swap the data between the two bucket arrays.

ANS:

```

1  // Exercise 25.7 Solution: BucketSort.cs
2  // Sort an array's values into ascending order using bucket sort.
3  using System;
4
5  public class BucketSort
6  {
7      private int[] data; // array of values
8      private static Random generator = new Random();
9
10     // create array of given size and fill with random integers
11     public BucketSort( int size )
12     {

```

```

13     data = new int[ size ]; // create space for array
14
15     // fill array with random ints in range 10-99
16     for ( int i = 0; i < size; i++ )
17         data[ i ] = generator.Next( 10, 100 );
18 } // end BucketSort constructor
19
20 // perform bucket sort algorithm on array
21 public void Sort()
22 {
23     // store maximum number of digits in numbers to sort
24     int totalDigits = NumberOfDigits();
25
26     // bucket array where numbers will be placed
27     int[ , ] pail = new int[ 10, data.Length + 1 ];
28
29     // go through all digit places and sort each number
30     // according to digit place value
31     for ( int pass = 1; pass <= totalDigits; pass++ )
32     {
33         DistributeElements( pail, pass ); // distribution pass
34         CollectElements( pail ); // gathering pass
35
36         if ( pass != totalDigits )
37             EmptyBucket( pail ); // set size of buckets to 0
38     } // end for
39 } // end method Sort
40
41 // determine number of digits in the largest number
42 public int NumberOfDigits()
43 {
44     int largest = data[ 0 ]; // set largest to first element
45
46     // loop over elements to find largest
47     foreach ( var element in data )
48         if ( element > largest )
49             largest = element; // set largest to current element
50
51     // calculate number of digits in largest value
52     int digits = ( int ) ( Math.Floor( Math.Log10( largest ) ) + 1 );
53
54     return digits;
55 } // end method NumberOfDigits
56
57 // distribute elements into buckets based on specified digit
58 public void DistributeElements( int[ , ] pail, int digit )
59 {
60     int bucketNumber; // number of bucket to place element
61     int elementNumber; // location in bucket to place element
62
63     // determine the divisor used to get specific digit
64     int divisor = ( int ) ( Math.Pow( 10, digit ) );
65

```

```

66     foreach ( var element in data )
67     {
68         // bucketNumber example for hundreds digit:
69         // ( 1234 % 1000 ) / 100 --> 2
70         bucketNumber = ( element % divisor ) / ( divisor / 10 );
71
72         // retrieve value in pail[ bucketNumber , 0 ] to
73         // determine the location in row to store element
74         elementNumber = ++pail[ bucketNumber, 0 ];
75         pail[ bucketNumber, elementNumber ] = element;
76     } // end foreach
77 } // end method DistributeElements
78
79 // return elements to original array
80 public void CollectElements( int[ , ] pails )
81 {
82     int subscript = 0; // initialize location in data
83
84     for ( int i = 0; i < 10; i++ ) // loop over buckets
85     {
86         // loop over elements in each bucket
87         for ( int j = 1; j <= pails[ i, 0 ]; j++ )
88             data[ subscript++ ] = pails[ i, j ]; // add element to array
89     } // end outer for
90 } // end method CollectElements
91
92 // set size of all buckets to zero
93 public void EmptyBucket( int[ , ] pails )
94 {
95     for ( int i = 0; i < 10; i++ )
96         pails[ i, 0 ] = 0; // set size of bucket to 0
97 } // end method EmptyBucket
98
99 // method to output values in array
100 public override string ToString()
101 {
102     string temporary = string.Empty;
103
104     // iterate through array
105     foreach ( var element in data )
106         temporary += element + " ";
107
108     temporary += "\n"; // add newline character
109     return temporary;
110 } // end method ToString
111 } // end class BucketSort

```

```

1 // Exercise 25.7 Solution: BucketSortTest.cs
2 // Test the bucket sort class.
3 using System;
4
5 public class BucketSortTest
6 {

```

```

7  public static void Main( string[] args )
8  {
9      // create object to perform bucket sort
10     BucketSort sortArray = new BucketSort( 10 );
11
12     Console.WriteLine( "Before:" );
13     Console.WriteLine( sortArray ); // display unsorted array
14
15     sortArray.Sort(); // sort array
16
17     Console.WriteLine( "After:" );
18     Console.WriteLine( sortArray ); // display sorted array
19 } // end Main
20 } // end class BucketSortTest

```

Before:
60 46 21 36 10 69 70 85 29 99

After:
10 21 29 36 46 60 69 70 85 99

20.8 (*Recursive Linear Search*) Modify Fig. 25.2 to use recursive method `RecursiveLinearSearch` to perform a linear search of the array. The method should receive the search key and starting index as arguments. If the search key is found, return its index in the array; otherwise, return -1. Each call to the recursive method should check one index in the array.

ANS:

```

1  // Exercise 25.8 Solution: LinearArray.cs
2  // Class that contains an array of random integers and a method
3  // that will search that array sequentially.
4  using System;
5
6  public class LinearArray
7  {
8      private int[] data; // array of values
9      private static Random generator = new Random();
10
11     // create array of given size and fill with random integers
12     public LinearArray( int size )
13     {
14         data = new int[ size ]; // create space for array
15
16         // fill array with random ints in range 10-99
17         for ( int i = 0; i < size; i++ )
18             data[ i ] = generator.Next( 10, 100 );
19     } // end LinearArray constructor
20
21     // perform a linear search on the data
22     public int LinearSearch( int search )
23     {
24         return RecursiveLinearSearch( search, 0 );
25     } // end method LinearSearch

```

```

26
27     public int RecursiveLinearSearch( int search, int start )
28     {
29         int location; // variable to store return value
30
31         if ( start >= data.Length ) // if at end of array
32             location = -1; // value not found
33         else
34         {
35             // if item is equal to search key
36             if ( data[ start ] == search )
37                 location = start; // return current location
38             else
39                 // recursively search rest of array
40                 location = RecursiveLinearSearch( search, start + 1 );
41         } // end else
42
43         return location; // return location of search key
44     } // end method RecursiveLinearSearch
45
46     // method to output values in array
47     public override string ToString()
48     {
49         string temporary = string.Empty;
50
51         // iterate through array
52         foreach ( var element in data )
53             temporary += element + " ";
54
55         temporary += "\n"; // add newline character
56         return temporary;
57     } // end method ToString
58 } // end class LinearArray

```

```

1 // Exercise 25.8 Solution: LinearSearchTest.cs
2 // Sequentially search an array for an item.
3 using System;
4
5 public class LinearSearchTest
6 {
7     public static void Main( string[] args )
8     {
9         int searchInt; // search
10        int position; // location of search key in array
11
12        // create array and output it
13        LinearArray searchArray = new LinearArray( 10 );
14        Console.WriteLine( searchArray );
15
16        // get first int input from user
17        Console.Write( "Please enter an integer value (-1 to quit): " );
18        searchInt = Convert.ToInt32( Console.ReadLine() );
19

```

```

20      // repeatedly input an integer; -1 will quit the application
21      while ( searchInt != -1 )
22      {
23          // search array linearly
24          position = searchArray.LinearSearch( searchInt );
25
26          // return value of -1 indicates integer was not found
27          if ( position == -1 )
28              Console.WriteLine( "The integer {0} was not found.\n",
29                                  searchInt );
30          else
31              Console.WriteLine(
32                  "The integer {0} was found in position {1}.\n",
33                  searchInt, position );
34
35          // get next int input from user
36          Console.Write( "Please enter an integer value (-1 to quit): " );
37          searchInt = Convert.ToInt32( Console.ReadLine() );
38      } // end while
39  } // end Main
40 } // end class LinearSearchTest

```

```
64 21 91 20 12 53 29 83 30 57
```

```
Please enter an integer value (-1 to quit): 12
The integer 12 was found in position 4.
```

```
Please enter an integer value (-1 to quit): 65
The integer 65 was not found.
```

```
Please enter an integer value (-1 to quit): -1
```

20.9 (*Recursive Binary Search*) Modify Fig. 25.4 to use recursive method `RecursiveBinarySearch` to perform a binary search of the array. The method should receive the search key, starting index and ending index as arguments. If the search key is found, return its index in the array. If the search key is not found, return -1.

ANS:

```

1  // Exercise 25.9 Solution: BinaryArray.cs
2  // Class that contains an array of random integers and a method
3  // that uses binary search to find an integer.
4  using System;
5
6  public class BinaryArray
7  {
8      private int[] data; // array of values
9      private static Random generator = new Random();
10
11     // create array of given size and fill with random integers
12     public BinaryArray( int size )
13     {
14         data = new int[ size ]; // create space for array

```

```
15
16     // fill array with random ints in range 10-99
17     for ( int i = 0; i < size; i++ )
18         data[ i ] = generator.Next( 10, 100 );
19
20     Array.Sort( data );
21 } // end BinaryArray constructor
22
23 // perform a binary search on the data
24 public int BinarySearch( int searchElement )
25 {
26     int low = 0; // low end of the search area
27     int high = data.Length - 1; // high end of the search area
28
29     return RecursiveBinarySearch( searchElement, low, high );
30 } // end method BinarySearch
31
32 public int RecursiveBinarySearch(
33     int searchElement, int low, int high )
34 {
35     if ( low > high ) // test base case; no element left to check
36         return -1;
37
38     // middle of the search area; element we will test first
39     int middle = ( low + high + 1 ) / 2;
40
41     // display remaining elements of array
42     Console.Write( RemainingElements( low, high ) );
43
44     // output spaces for alignment
45     for ( int i = 0; i < middle; i++ )
46         Console.Write( "  " );
47
48     Console.WriteLine( " * " ); // indicate current middle
49
50     // variable to return; -1 if the value was not found
51     int location = -1;
52
53     // if the element is found
54     if ( searchElement == data[ middle ] )
55         location = middle; // location is the current middle
56     // middle element is too high
57     else if ( searchElement < data[ middle ] )
58         // eliminate the higher half
59         location = RecursiveBinarySearch( searchElement,
60             low, middle - 1 );
61     else // middle element is too low
62         // eliminate the lower half
63         location = RecursiveBinarySearch( searchElement,
64             middle + 1, high );
65
66     return location; // return location of search key
67 } // end method RecursiveBinarySearch
68
```

```

69 // method to output certain values in array
70 public string RemainingElements( int low, int high )
71 {
72     string temporary = string.Empty;
73
74     // output spaces for alignment
75     for ( int i = 0; i < low; i++ )
76         temporary += " ";
77
78     // output elements left in array
79     for ( int i = low; i <= high; i++ )
80         temporary += data[ i ] + " ";
81
82     temporary += "\n";
83     return temporary;
84 } // end method RemainingElements
85
86 // method to output values in array
87 public override string ToString()
88 {
89     return RemainingElements( 0, data.Length - 1 );
90 } // end method ToString
91 } // end class BinaryArray

```

```

30         else
31             Console.WriteLine(
32                 "The integer {0} was found in position {1}.\n",
33                 searchInt, position );
34
35             // get next int input from user
36             Console.Write( "Please enter an integer value (-1 to quit): " );
37             searchInt = Convert.ToInt32( Console.ReadLine() );
38         } // end while
39     } // end Main
40 } // end class BinarySearchTest

```

```
18 19 20 26 30 40 47 49 64 64 67 76 82 84 88 95
```

```
Please enter an integer value (-1 to quit): 26
```

```
18 19 20 26 30 40 47 49 64 64 67 76 82 84 88 95
                        *
```

```
18 19 20 26 30 40 47 49
                        *
```

```
18 19 20 26
```

```
        *
```

```
        26
```

```
        *
```

```
The integer 26 was found in position 3.
```

```
Please enter an integer value (-1 to quit): -1
```

20.10 (*Quicksort*) The recursive sorting technique called quicksort uses the following basic algorithm for a one-dimensional array of values:

- a) *Partitioning Step*: Take the first element of the unsorted array and determine its final location in the sorted array (i.e., all values to the left of the element in the array are less than the element, and all values to the right of the element in the array are greater than the element—we show how to do this below). We now have one element in its proper location and two unsorted subarrays.
- b) *Recursive Step*: Perform *Step 1* on each unsorted subarray.

Each time *Step 1* is performed on a subarray, another element is placed in its final location in the sorted array, and two unsorted subarrays are created. When a subarray consists of one element, that element is in its final location (because a one-element array is already sorted).

The basic algorithm seems simple enough, but how do we determine the final position of the first element of each subarray? As an example, consider the following set of values (the element in bold is the partitioning element—it will be placed in its final location in the sorted array):

```
37 2 6 4 89 8 10 12 68 45
```

- a) Starting from the rightmost element of the array, compare each element with 37 until an element less than 37 is found, then swap 37 and that element. The first element less than 37 is 12, so 37 and 12 are swapped. The new array is

```
12 2 6 4 89 8 10 37 68 45
```

Element 12 is in italics to indicate that it was just swapped with 37.

- b) Starting from the left of the array, but beginning with the element after 12, compare each element with 37 until an element greater than 37 is found—then swap 37 and that

element. The first element greater than 37 is 89, so 37 and 89 are swapped. The new array is

12 2 6 4 37 8 10 89 68 45

- c) Starting from the right, but beginning with the element before 89, compare each element with 37 until an element less than 37 is found—then swap 37 and that element. The first element less than 37 is 10, so 37 and 10 are swapped. The new array is

12 2 6 4 10 8 37 89 68 45

- d) Starting from the left, but beginning with the element after 10, compare each element with 37 until an element greater than 37 is found—then swap 37 and that element. There are no more elements greater than 37, so when we compare 37 with itself, we know that 37 has been placed in its final location of the sorted array. Every value to the left of 37 is smaller than it, and every value to the right of 37 is larger than it.

Once the partition has been applied on the previous array, there are two unsorted subarrays. The subarray with values less than 37 contains 12, 2, 6, 4, 10 and 8. The subarray with values greater than 37 contains 89, 68 and 45. The sort continues recursively, with both subarrays being partitioned in the same manner as the original array.

Based on the preceding discussion, write recursive method `QuickSortHelper` to sort a one-dimensional integer array. The method should receive as arguments a starting index and an ending index in the original array being sorted.

ANS:

```

1 // Exercise 25.10 Solution: QuickSort.cs
2 // Class that creates an integer array filled with random
3 // values and can sort that array using the quicksort algorithm
4 using System;
5
6 public class QuickSort
7 {
8     private int[] data; // array of values
9     private static Random generator = new Random();
10
11     // create array of given size and fill with random integers
12     public QuickSort( int size )
13     {
14         data = new int[ size ]; // create space for array
15
16         // fill array with random ints in range 10-99
17         for ( int i = 0; i < size; i++ )
18             data[ i ] = generator.Next( 10, 100 );
19     } // end QuickSort constructor
20
21     // call recursive method QuickSortHelper
22     public void Sort()
23     {
24         QuickSortHelper( 0, data.Length - 1 );
25     } // end method Sort
26
27     // recursive method to sort array using quicksort
28     public void QuickSortHelper( int left, int right )
29     {

```

```

30     int pivot = left;
31     int edge = right;
32     int swapIndex = -1; // index of the element to swap with pivot
33
34     // loop until the edge index is equal to the pivot index
35     while ( pivot != edge )
36     {
37         // if pivot index is less than edge index, check right side
38         if ( pivot < edge )
39         {
40             // if there is an element to the left less than pivot
41             if ( ( swapIndex = FindRight( pivot, edge ) ) >= 0 )
42             {
43                 Swap( swapIndex, pivot ); // swap the element with pivot
44
45                 // swap the index of pivot and edge, add 1 to the edge
46                 // so the element is not considered again
47                 edge = pivot + 1;
48                 pivot = swapIndex;
49
50                 swapIndex = -1; // reset the swapIndex
51             } // end if
52             else
53                 break; // break out of the while statement
54         } // end if
55         else // pivot index is greater than edge index, check left side
56         {
57             // if there is an element to the right greater than pivot
58             if ( ( swapIndex = FindLeft( pivot, edge ) ) >= 0 )
59             {
60                 Swap( swapIndex, pivot ); // swap the element with pivot
61
62                 // swap the index of pivot and edge, add 1 to the edge
63                 // so the item is not considered again
64                 edge = pivot - 1;
65                 pivot = swapIndex;
66
67                 swapIndex = -1; // reset the swapIndex
68             } // end if
69             else
70                 break; // break out of the while statement
71         } // end else
72     } // end while
73
74     if ( left < pivot - 1 ) // if more than one element on left
75         QuickSortHelper( left, pivot - 1 ); // sort left side
76
77     if ( pivot + 1 < right ) // if more than one element on right
78         QuickSortHelper( pivot + 1, right ); // sort right side
79 } // end method quicksortHelper
80
81 // helper method to swap values in two elements
82 public void Swap( int first, int second )
83 {


```

```
84     int temporary = data[ first ]; // store first in temporary
85     data[ first ] = data[ second ]; // replace first with second
86     data[ second ] = temporary; // put temporary in second
87 } // end method Swap
88
89 // helper method to find the first element less than the pivot
90 // starting from the right and moving left; return -1 if none found
91 private int FindRight( int pivot, int edge )
92 {
93     int index = -1;
94
95     for ( int i = edge; i > pivot; i-- )
96     {
97         if ( data[ i ] < data[ pivot ] )
98         {
99             index = i;
100             break;
101         } // end if
102     } // end for
103
104     return index; // return the index of the selected element
105 } // end method FindRight
106
107 // helper method to find the first element greater than the pivot
108 // starting from the left and moving right; return -1 if none found
109 private int FindLeft( int pivot, int edge )
110 {
111     int index = -1;
112
113     for ( int i = edge; i < pivot; i++ )
114     {
115         if ( data[ i ] > data[ pivot ] )
116         {
117             index = i;
118             break;
119         } // end if
120     } // end for
121
122     return index; // return the index of the selected element
123 } // end method FindLeft
124
125 // method to output values in array
126 public override string ToString()
127 {
128     string temporary = string.Empty;
129
130     // iterate through array
131     foreach ( var element in data )
132         temporary += element + " ";
133
134     temporary += "\n"; // add newline character
135     return temporary;
136 } // end method ToString
137 } // end class QuickSort
```

```
1 // Exercise 25.10 Solution: QuickSortTest.cs
2 // Test the quicksort class.
3 using System;
4
5 public class QuickSortTest
6 {
7     public static void Main( string[] args )
8     {
9         // create object to perform selection sort
10        QuickSort sortArray = new QuickSort( 10 );
11
12        Console.WriteLine( "Before:" );
13        Console.WriteLine( sortArray ); // display unsorted array
14
15        sortArray.Sort(); // sort array
16
17        Console.WriteLine( "After:" );
18        Console.WriteLine( sortArray ); // display sorted array
19    } // end Main
20 } // end class QuickSortTest
```

Before:
29 65 37 82 38 91 87 90 36 74

After:
29 36 37 38 65 74 82 87 90 91



*Much that I bound,
I could not free;
Much that I freed
returned to me.*

—Lee Wilson Dodd

There is always room at the top.

—Daniel Webster

*I think that I shall never see
A poem lovely as a tree.*

—Joyce Kilmer

Objectives

In this chapter you'll learn:

- To form linked data structures using references, self-referential classes and recursion.
- How boxing and unboxing enable simple-type values to be used where **objects** are expected in a program.
- To create and manipulate dynamic data structures, such as linked lists, queues, stacks and binary trees.
- Various important applications of linked data structures.
- To create reusable data structures with classes, inheritance and composition.

Self-Review Exercises

21.1 State whether each of the following is *true* or *false*. If *false*, explain why.

a) In a queue, the first item to be added is the last item to be removed.

ANS: False. A queue is a first-in, first-out data structure—the first item added is the first item removed.

b) Trees can have no more than two child nodes per node.

ANS: False. In general, trees may have as many child nodes per node as is necessary. Only binary trees are restricted to no more than two child nodes per node.

c) A tree node with no children is called a leaf node.

ANS: True.

d) Linked-list nodes are stored contiguously in memory.

ANS: False. Linked-list nodes are logically contiguous, but they need not be stored in a physically contiguous memory space.

e) The primary operations of the stack data structure are enqueue and dequeue.

ANS: False. Those are the primary operations of a queue. The primary operations of a stack are push and pop.

f) Lists, stacks and queues are linear data structures.

ANS: True.

21.2 Fill in the blanks in each of the following statements:

a) A(n) _____ class is used to define nodes that form dynamic data structures, which can grow and shrink at execution time.

ANS: self-referential.

b) Operator _____ allocates memory dynamically; this operator returns a reference to the allocated memory.

ANS: new.

c) A(n) _____ is a constrained version of a linked list in which nodes can be inserted and deleted only from the start of the list; this data structure returns node values in last-in, first-out order.

ANS: stack.

d) A queue is a(n) _____ data structure, because the first nodes inserted are the first nodes removed.

ANS: first-in, first-out (FIFO).

e) A(n) _____ is a constrained version of a linked list in which nodes can be inserted only at the end of the list and deleted only from the start of the list.

ANS: queue.

f) A(n) _____ is a nonlinear, two-dimensional data structure that contains nodes with two or more links.

ANS: tree.

g) The nodes of a(n) _____ tree contain two link members.

ANS: binary.

h) The tree-traversal algorithm that processes the node then processes all the nodes to its left followed by all the nodes to its right is called _____.

ANS: preorder.

Exercises

21.3 (*Merging Ordered-List Objects*) Write a program that merges two ordered-list objects of integers into a single ordered-list object of integers. Method Merge of class ListMerge should receive references to each of the list objects to be merged and should return a reference to the merged-list object.

ANS:

```

1  // Ex. 21.3: ListMerge.cs
2  // Has a method for merging two lists.
3  using System;
4  using LinkedListLibrary;
5
6  public class ListMerge
7  {
8      // constructor
9      public ListMerge()
10     {
11     } // end constructor
12
13     // merges two ordered linked-lists
14     public static List Merge( List first, List second )
15     {
16         List newList = new List();
17         int one = 0, two = 0; // these store values removed from each list
18         int choice = -1; // determines which list to remove from
19
20         // while there is something in each list compare
21         // the first element of both, add the smallest one,
22         // do this until one of the lists is empty
23         while ( ( !first.IsEmpty() || choice == 2 ) &&
24             ( !second.IsEmpty() || choice == 1 ) )
25         {
26             switch ( choice )
27             {
28                 case -1:
29                     one = ( int ) first.RemoveFromFront();
30                     two = ( int ) second.RemoveFromFront();
31                     break;
32                 case 1:
33                     one = ( int ) first.RemoveFromFront();
34                     break;
35                 case 2:
36                     two = ( int ) second.RemoveFromFront();
37                     break;
38             } // end switch
39
40             if ( one < two )
41             {
42                 newList.InsertAtBack( one );
43                 choice = 1;
44             } // end if
45             else
46             {
47                 newList.InsertAtBack( two );

```

```
48         choice = 2;
49     } // end else
50 } // end while
51
52 // insert the element that still resides in one or two
53 if ( choice == 1 )
54     newList.InsertAtBack( two );
55 else if ( choice == 2 )
56     newList.InsertAtBack( one );
57
58 // get non-empty list
59 List leftOver = ( first.IsEmpty() ? second : first );
60
61 // get remaining elements
62 while ( !leftOver.IsEmpty() )
63     newList.InsertAtBack( leftOver.RemoveFromFront() );
64
65 return newList;
66 } // end method Merge
67
68 // test Merge method
69 public static void Main( string[] args )
70 {
71     List firstList = new List();
72     List secondList = new List();
73
74     // fill lists with integers
75     firstList.InsertAtBack( 3 );
76     firstList.InsertAtBack( 4 );
77     firstList.InsertAtBack( 7 );
78     firstList.InsertAtBack( 9 );
79     firstList.InsertAtBack( 12 );
80     firstList.InsertAtBack( 16 );
81     firstList.InsertAtBack( 18 );
82     firstList.InsertAtBack( 23 );
83     firstList.InsertAtBack( 33 );
84     secondList.InsertAtBack( 5 );
85     secondList.InsertAtBack( 11 );
86     secondList.InsertAtBack( 17 );
87     secondList.InsertAtBack( 22 );
88     secondList.InsertAtBack( 25 );
89     secondList.InsertAtBack( 28 );
90
91     // display lists before merging
92     firstList.Display();
93     secondList.Display();
94
95     // display result of merging
96     Console.WriteLine( "\nThe merged list:" );
97     List mergedList = Merge( firstList, secondList );
98     mergedList.Display();
99 } // end Main
100 } // end class ListMerge
```

The list is: 3 4 7 9 12 16 18 23 33

The list is: 5 11 17 22 25 28

The merged list:

The list is: 3 4 5 7 9 11 12 16 17 18 22 23 25 28 33

21.4 (*Reversing a Line of Text with a Stack*) Write a program that inputs a line of text and uses a stack object to display the line reversed.

ANS:

```

1  // Ex. 21.4: ReverseString.cs
2  // Program that uses a stack to reverse a string.
3  using System;
4  using StackInheritanceLibrary;
5
6  class ReverseString
7  {
8      public static void Main( string[] args )
9      {
10         Console.Write( "Please enter word to reverse: " );
11
12         // get input
13         string word = Console.ReadLine();
14         StackInheritance stack = new StackInheritance();
15
16         // push each character onto stack
17         foreach ( var character in word )
18             stack.Push( character );
19
20         Console.Write( "The word reversed is: " );
21
22         // pop each character off stack
23         for ( int i = 0; i < word.Length; i++ )
24             Console.Write( stack.Pop() );
25
26         Console.WriteLine();
27     } // end Main
28 } // end class ReverseString

```

Please enter word to reverse: **deitel**
The word reversed is: **letied**

21.5 (*Palindromes*) Write a program that uses a stack to determine whether a string is a palindrome (i.e., the string is spelled identically backward and forward). The program should ignore capitalization, spaces and punctuation.

ANS:

```

1  // Ex. 21.5: Palindrome.cs
2  // Tests for a palindrome using a stack.
3  using System;
4  using StackInheritanceLibrary;
5  using System.Text;
6
7  class Palindrome
8  {
9      public static void Main( string[] args )
10     {
11         Console.Write( "Please enter a word: " );
12
13         string word = Console.ReadLine();
14         StringBuilder testPalindrome = new StringBuilder( word );
15         int i;
16
17         for ( i = 0; i < testPalindrome.Length; i++ )
18         {
19             if ( !char.IsLetterOrDigit( testPalindrome[ i ] ) )
20             {
21                 testPalindrome.Remove( i, 1 );
22                 --i;
23             } // end if
24         } // end for
25
26         int halfWay = ( int ) Math.Floor(
27             testPalindrome.Length / 2.0 ) - 1;
28
29         StackInheritance stack = new StackInheritance();
30
31         // push half of word onto stack
32         for ( i = 0; i <= halfWay; i++ )
33             stack.Push( Char.ToLower( testPalindrome[ i ] ) );
34
35         // if odd length, increase i by 1
36         if ( halfWay == ( ( testPalindrome.Length - 1 ) / 2 ) - 1 )
37             ++i;
38
39         // compare what is on stack to rest of word
40         for ( int j = i; j < testPalindrome.Length; j++ )
41         {
42             if ( Char.ToLower( testPalindrome[ j ] ) !=
43                 ( char ) stack.Pop() )
44             {
45                 Console.WriteLine( word + " is not a palindrome" );
46
47                 return;
48             } // end if
49         } // end for

```

```

50
51     Console.WriteLine( word + " is a palindrome" );
52 } // end Main
53 } // end class Palindrome

```

```

Please enter a word: deitel
deitel is not a palindrome

```

```

Please enter a word: able was i ere i saw elba
able was i ere i saw elba is a palindrome

```

21.6 (Evaluating Expressions with a Stack) Stacks are used by compilers to evaluate expressions and generate machine-language code. In this and the next exercise, we investigate how compilers evaluate arithmetic expressions consisting only of constants, operators and parentheses.

Humans generally write expressions like $3 + 4$ and $7 / 9$, in which the operator (+ or / here) is written between its operands—this is called *infix notation*. Computers “prefer” *postfix notation*, in which the operator is written to the right of its two operands. The preceding infix expressions would appear in postfix notation as $3\ 4\ +$ and $7\ 9\ /$, respectively.

To evaluate a complex infix expression, a compiler would first convert the expression to postfix notation, then evaluate the postfix version of the expression. Each of these algorithms requires only a single left-to-right pass of the expression. Each algorithm uses a stack object in support of its operation, and in each algorithm the stack is used for a different purpose.

Here, you’ll implement the infix-to-postfix conversion algorithm. In the next exercise, you’ll implement the postfix-expression evaluation algorithm. In a later exercise, you’ll discover that code you write in this exercise can help you implement a complete working compiler.

Write class `InfixToPostfixConverter` to convert an ordinary infix arithmetic expression (assume a valid expression is entered), with single-digit integers, such as

$(6 + 2) * 5 - 8 / 4$

to a postfix expression. The postfix version of the preceding infix expression is

$6\ 2\ +\ 5\ * \ 8\ 4\ /\ -$

The program should read the expression into `StringBuilder infix`, then use class `StackInheritance` (implemented in Fig. 26.13) to help create the postfix expression in `StringBuilder postfix`. The algorithm for creating a postfix expression is as follows:

- a) Push a left parenthesis '(' on the stack.
- b) Append a right parenthesis ')' to the end of `infix`.
- c) While the stack is not empty, read `infix` from left to right and do the following:
 - If the current character in `infix` is a digit, append it to `postfix`.
 - If the current character in `infix` is a left parenthesis, push it onto the stack.
 - If the current character in `infix` is an operator:
 - Pop operators (if there are any) at the top of the stack while they have equal or higher precedence than the current operator, and append the popped operators to `postfix`.
 - Push the current character in `infix` onto the stack.
 - If the current character in `infix` is a right parenthesis:
 - Pop operators from the top of the stack and append them to `postfix` until a left parenthesis is at the top of the stack.
 - Pop (and discard) the left parenthesis from the stack.

The following arithmetic operations are allowed in an expression:

- + addition
- subtraction
- * multiplication
- / division
- ^ exponentiation
- % modulus

Some of the methods you may want to provide in your program follow:

- a) Method `ConvertToPostfix`, which converts the infix expression to postfix notation.
- b) Method `IsOperator`, which determines whether `c` is an operator.
- c) Method `Precedence`, which determines whether the precedence of `operator1` (from the infix expression) is less than, equal to or greater than the precedence of `operator2` (from the stack). The method returns `true` if `operator1` has lower precedence than or equal precedence to `operator2`. Otherwise, `false` is returned.

ANS:

```

1 // Ex. 21.6: InfixToPostfixConverter.cs
2 // Converts an expression from infix to postfix.
3 using System;
4 using System.Text;
5 using StackInheritanceLibrary;
6
7 public class InfixToPostfixConverter
8 {
9     // determines whether a char is a valid operator
10    public static bool IsOperator( char character )
11    {
12        if ( character == '+' || character == '-' || character == '*' ||
13            character == '/' || character == '^' || character == '%' )
14            return true;
15        else
16            return false;
17    } // end method IsOperator
18
19    // determines whether operator2 has higher precedence than operator1
20    public static bool Precedence( char operator1, char operator2 )
21    {
22        if ( operator2 == '^' || operator2 == '%' )
23            return true;
24
25        if ( operator1 == '^' || operator1 == '%' )
26            return false;
27
28        if ( operator2 == '*' || operator2 == '/' )
29            return true;
30
31        if ( operator1 == '*' || operator1 == '/' )
32            return false;
33
34        return true;
35    } // end method Precedence
36

```

```

37 // convert a string representing an infix expression to postfix
38 public static string ConvertToPostfix( string expression )
39 {
40     StackInheritance stack = new StackInheritance();
41     StringBuilder postfix = new StringBuilder( string.Empty );
42     StringBuilder infix = new StringBuilder( expression );
43     int i = 0;
44
45     stack.Push( '(' ); // step (a)
46     infix.Append( ")" ); // step (b)
47
48     while ( !stack.IsEmpty() ) // step (c)
49     {
50         Console.WriteLine( "i = " + i +
51             "    Stack contains: " );
52         stack.Display();
53         Console.WriteLine();
54
55         // if it is a digit, add it to output
56         if ( Char.IsDigit( infix[ i ] ) )
57             postfix.Append( infix[ i ] + " " );
58
59         // if it is a left parenthesis, push onto stack
60         if ( infix[ i ] == '(' )
61             stack.Push( '(' );
62
63         // if it is an operator
64         if ( InfixToPostfixConverter.IsOperator( infix[ i ] ) )
65         {
66             // pop and add to output all operators of higher or
67             // equal precedence
68             while ( !stack.IsEmpty() )
69             {
70                 char character = ( char ) stack.Pop();
71
72                 // if the top of the stack is an operator
73                 if ( IsOperator( character ) )
74                 {
75                     // if it is of higher or equal precedence add to output
76                     if ( Precedence( infix[ i ], character ) )
77                         postfix.Append( character + " " );
78                     else
79                     {
80                         // otherwise push back on stack and leave loop
81                         stack.Push( character );
82                         break;
83                     } // end else
84                 } // end if
85                 else
86                 {
87                     // otherwise, push back on stack, and leave loop
88                     stack.Push( character );
89                     break;
90                 } // end else
91             } // end while

```

```

92
93         // push current operator on stack
94         stack.Push( infix[ i ] );
95     } // end if
96
97     // if it is a right parenthesis
98     if ( infix[ i ] == ')' )
99     {
100         // add all operators on stack to output
101         while ( !stack.IsEmpty() )
102         {
103             char character = ( char ) stack.Pop();
104
105             if ( character != '(' )
106                 postfix.Append( character + " " );
107             else
108                 break;
109         } // end while
110     } // end if
111
112     // increment i
113     ++i;
114 } // end while
115
116     return postfix.ToString();
117 } // end method ConvertToPostfix
118
119 // test ConvertToPostfix method
120 public static void Main( string[] args )
121 {
122     Console.Write( "Please enter an expression in infix notation: " );
123
124     string expression = Console.ReadLine();
125     string postfix =
126         InfixToPostfixConverter.ConvertToPostfix( expression );
127
128     Console.WriteLine(
129         "The expression in postfix notation is: " + postfix );
130 } // end Main
131 } // end class InfixToPostfixConverter

```

```

Please enter an expression in infix notation: 2+3*5
i = 0   Stack contains:
The stack is: (

```

```

i = 1   Stack contains:
The stack is: (

```

```

i = 2   Stack contains:
The stack is: + (

```

```
i = 3   Stack contains:
The stack is: + (
```

```
i = 4   Stack contains:
The stack is: * + (
```

```
i = 5   Stack contains:
The stack is: * + (
```

```
The expression in postfix notation is: 2 3 5 * +
```

21.7 (*Evaluating a Postfix Expression with a Stack*) Write class `PostfixEvaluator`, which evaluates a postfix expression (assume it is valid) such as

```
6 2 + 5 * 8 4 / -
```

The program should read a postfix expression consisting of digits and operators into a `StringBuilder`. Using the stack class from Exercise 21.6, the program should scan the expression and evaluate it. The algorithm (for single-digit numbers) is as follows:

- Append a right parenthesis ')' to the end of the postfix expression. When the right-parenthesis character is encountered, no further processing is necessary.
- When the right-parenthesis character has not been encountered, read the expression from left to right.

If the current character is a digit, do the following:

Push its integer value on the stack (the integer value of a digit character is its value in the computer's character set minus the value of '0' in Unicode).

Otherwise, if the current character is an *operator*:

Pop the two top elements of the stack into variables *x* and *y*.

Calculate *y operator x*.

Push the result of the calculation onto the stack.

- When the right parenthesis is encountered in the expression, pop the top value of the stack. This is the result of the postfix expression.

[*Note:* In *Part b* above (based on the sample expression at the beginning of this exercises), if the operator is '/', the top of the stack is 4 and the next element in the stack is 8, then pop 4 into *x*, pop 8 into *y*, evaluate 8 / 4 and push the result, 2, back on the stack. This note also applies to operator '-'.] The arithmetic operations allowed in an expression are:

- + addition
- subtraction
- * multiplication
- / division
- ^ exponentiation
- % modulus

You may want to provide the following methods:

- Method `EvaluatePostfixExpression`, which evaluates the postfix expression.
- Method `Calculate`, which evaluates the expression *op1 operator op2*.

ANS:

```

1 // Ex. 21.7: PostfixEvaluator.cs
2 // Evaluates a postfix expression.
3 using System;
4 using StackInheritanceLibrary;
5
6 public class PostfixEvaluator
7 {
8     // constructor
9     public PostfixEvaluator()
10    {
11    } // end constructor
12
13    public static int EvaluatePostfixExpression( string expression )
14    {
15        expression += ")"; // step (a)
16
17        int i = 0;
18        StackInheritance stack = new StackInheritance();
19
20        while ( expression[ i ] != ')' ) // step (b)
21        {
22            if ( Char.IsDigit( expression[ i ] ) )
23                stack.Push( expression[ i ] - '0' );
24            else if ( !Char.IsWhiteSpace( expression[ i ] ) )
25            {
26                int x = ( int ) stack.Pop();
27                int y = ( int ) stack.Pop();
28
29                stack.Push( Calculate( y, x, expression[ i ] ) );
30            } // end else
31
32            ++i;
33        } // end while
34
35        return ( int ) stack.Pop(); // step (c)
36    } // end method EvaluatePostfixExpression
37
38    // perform an operation on the two operands
39    public static int Calculate(
40        int operand1, int operand2, char operation )
41    {
42        int result = 0;
43
44        switch ( operation )
45        {
46            case '+':
47                result = operand1 + operand2;
48                break;
49            case '-':
50                result = operand1 - operand2;
51                break;

```

```

52         case '/':
53             result = operand1 / operand2;
54             break;
55         case '*':
56             result = operand1 * operand2;
57             break;
58         case '%':
59             result = operand1 % operand2;
60             break;
61         case '^':
62             result = ( int ) Math.Pow( operand1, operand2 );
63             break;
64     } // end switch
65
66     return result;
67 } // end method Calculate
68
69 // test EvaluatePostfixExpression
70 public static void Main( string[] args )
71 {
72     Console.Write(
73         "Please enter an expression in postfix notation: " );
74
75     string expression = Console.ReadLine();
76
77     Console.WriteLine( "The value of the expression is: " +
78         PostfixEvaluator.EvaluatePostfixExpression( expression ) );
79 } // end Main
80 } // end class PostfixEvaluator

```

```

Please enter an expression in postfix notation: 2 3 5 * +
The value of the expression is: 17

```

21.8 (*Level-Order Binary Tree-Traversal*) The program of Fig. 26.21 illustrated three recursive methods of traversing a binary tree—inorder, preorder, and postorder traversals. This exercise presents the *level-order traversal* of a binary tree, in which the node values are displayed level by level, starting at the root-node level. The nodes on each level are displayed from left to right. The level-order traversal is not a recursive algorithm. It uses a queue object to control the output of the nodes. The algorithm is as follows:

- a) Insert the root node in the queue.
- b) While there are nodes left in the queue, do the following:
 - Get the next node in the queue.
 - Display the node's value.
 - If the reference to the left child of the node is not null:
 - Insert the left child node in the queue.
 - If the reference to the right child of the node is not null:
 - Insert the right child node in the queue.

Write method `LevelOrderTraversal` to perform a level-order traversal of a binary-tree object. Modify the program of Fig. 26.21 to use this method. [*Note:* You also will need to use the queue-processing methods of Fig. 26.16 in this program.]

ANS:

```

1 // Ex. 21.8: BinaryTreeLibrary.cs
2 // Declaration of class TreeNode and class Tree.
3 using System;
4 using QueueInheritanceLibrary;
5
6 namespace BinaryTreeLibrary
7 {
8     // class TreeNode declaration
9     class TreeNode
10    {
11        // initialize data and make this a leaf node
12        public TreeNode( int nodeData )
13        {
14            Data = nodeData;
15            LeftNode = RightNode = null; // node has no children
16        } // end constructor
17
18        // auto-implemented property LeftNode
19        public TreeNode LeftNode { get; set; }
20
21        // auto-implemented property Data
22        public int Data { get; set; }
23
24        // auto-implemented property RightNode
25        public TreeNode RightNode { get; set; }
26
27        // insert TreeNode into Tree that contains nodes;
28        // ignore duplicate values
29        public void Insert( int insertValue )
30        {
31            if ( insertValue < Data ) // insert in left subtree
32            {
33                // insert new TreeNode
34                if ( LeftNode == null )
35                    LeftNode = new TreeNode( insertValue );
36                else // continue traversing left subtree
37                    LeftNode.Insert( insertValue );
38            } // end if
39            else if ( insertValue > Data ) // insert in right subtree
40            {
41                // insert new TreeNode
42                if ( RightNode == null )
43                    RightNode = new TreeNode( insertValue );
44                else // continue traversing right subtree
45                    RightNode.Insert( insertValue );
46            } // end else if
47        } // end method Insert
48    } // end class TreeNode
49
50    // class Tree declaration
51    public class Tree
52    {

```

```
53     private TreeNode root;
54
55     // construct an empty Tree of integers
56     public Tree()
57     {
58         root = null;
59     } // end constructor
60
61     // Insert a new node in the binary search tree.
62     // If the root node is null, create the root node here.
63     // Otherwise, call the insert method of class TreeNode.
64     public void InsertNode( int insertValue )
65     {
66         if ( root == null )
67             root = new TreeNode( insertValue );
68         else
69             root.Insert( insertValue );
70     } // end method InsertNode
71
72     // begin preorder traversal
73     public void PreorderTraversal()
74     {
75         PreorderHelper( root );
76     } // end method PreorderTraversal
77
78     // recursive method to perform preorder traversal
79     private void PreorderHelper( TreeNode node )
80     {
81         if ( node == null )
82             return;
83
84         // output node data
85         Console.Write( node.Data + " " );
86
87         // traverse left subtree
88         PreorderHelper( node.LeftNode );
89
90         // traverse right subtree
91         PreorderHelper( node.RightNode );
92     } // end method PreorderHelper
93
94     // begin inorder traversal
95     public void InorderTraversal()
96     {
97         InorderHelper( root );
98     } // end method InorderTraversal
99
100    // recursive method to perform inorder traversal
101    private void InorderHelper( TreeNode node )
102    {
103        if ( node == null )
104            return;
105    }
```

```
106         // traverse left subtree
107         InorderHelper( node.LeftNode );
108
109         // output node data
110         Console.Write( node.Data + " " );
111
112         // traverse right subtree
113         InorderHelper( node.RightNode );
114     } // end method InorderHelper
115
116     // begin postorder traversal
117     public void PostorderTraversal()
118     {
119         PostorderHelper( root );
120     } // end method PostorderTraversal
121
122     // recursive method to perform postorder traversal
123     private void PostorderHelper( TreeNode node )
124     {
125         if ( node == null )
126             return;
127
128         // traverse left subtree
129         PostorderHelper( node.LeftNode );
130
131         // traverse right subtree
132         PostorderHelper( node.RightNode );
133
134         // output node data
135         Console.Write( node.Data + " " );
136     } // end method PostorderHelper
137
138     // traverse the tree level-by-level
139     public void LevelOrderTraversal()
140     {
141         QueueInheritance queue = new QueueInheritance();
142         TreeNode current;
143
144         // enqueue root
145         queue.Enqueue( root );
146
147         // while queue is not empty
148         while ( !queue.IsEmpty() )
149         {
150             // dequeue an item
151             current = ( TreeNode ) queue.Dequeue();
152
153             // display item
154             Console.Write( current.Data + " " );
155
156             // if left node is not null, enqueue it
157             if ( current.LeftNode != null )
158                 queue.Enqueue( current.LeftNode );
159         }
```

```

160         // if right node is not null, enqueue it
161         if ( current.RightNode != null )
162             queue.Enqueue( current.RightNode );
163     } // end while
164 } // end method LevelOrderTraversal
165 } // end class Tree
166 } // end namespace BinaryTreeLibrary

```

```

1 // Ex. 21.8: TreeTest.cs
2 // This program tests class Tree.
3 using System;
4 using BinaryTreeLibrary;
5
6 // class TreeTest declaration
7 public class TreeTest
8 {
9     // test class Tree
10    public static void Main( string[] args )
11    {
12        Tree tree = new Tree();
13        int insertValue;
14
15        Console.WriteLine( "Inserting values: " );
16        Random random = new Random();
17
18        // insert 10 random integers from 0-99 in tree
19        for ( int i = 1; i <= 10; i++ )
20        {
21            insertValue = random.Next( 100 );
22            Console.Write( insertValue + " " );
23
24            tree.InsertNode( insertValue );
25        } // end for
26
27        // perform preorder traversal of tree
28        Console.WriteLine( "\n\nPreorder traversal" );
29        tree.PreorderTraversal();
30
31        // perform inorder traversal of tree
32        Console.WriteLine( "\n\nInorder traversal" );
33        tree.InorderTraversal();
34
35        // perform postorder traversal of tree
36        Console.WriteLine( "\n\nPostorder traversal" );
37        tree.PostorderTraversal();
38
39        // perform level order traversal of tree
40        Console.WriteLine( "\n\nLevel Order traversal" );
41        tree.LevelOrderTraversal();
42
43        Console.WriteLine();
44    } // end Main
45 } // end class TreeTest

```

Inserting values:

68 17 79 92 25 82 33 46 1 34

Preorder traversal

68 17 1 25 33 46 34 79 92 82

Inorder traversal

1 17 25 33 34 46 68 79 82 92

Postorder traversal

1 34 46 33 25 17 82 92 79 68

Level Order traversal

68 17 79 1 25 92 33 82 46 34



...our special individuality, as distinguished from our generic humanity.

—Oliver Wendell Holmes, Sr.

Every man of genius sees the world at a different angle from his fellows.

—Havelock Ellis

Born under one law, to another bound.

—Lord Brooke

Objectives

In this chapter you'll learn:

- To create generic methods that perform identical tasks on arguments of different types.
- To create a generic `Stack` class that can be used to store objects of any class or interface type.
- To understand how to overload generic methods with nongeneric methods or with other generic methods.
- To understand the `new()` constraint of a type parameter.
- To apply multiple constraints to a type parameter.

Self-Review Exercises

- 22.1** State whether each of the following is *true* or *false*. If *false*, explain why.
- a) A generic method cannot have the same method name as a non-generic method.
ANS: False. A generic method can be overloaded by non-generic methods with the same or a different number of arguments.
 - b) All generic method declarations have a type parameter list that immediately precedes the method name.
ANS: False. All generic method declarations have a type parameter list that immediately follows the method's name.
 - c) A generic method can be overloaded by another generic method with the same method name but a different number of type parameters.
ANS: True.
 - d) A type parameter can be declared only once in the type parameter list but can appear more than once in the method's parameter list.
ANS: True.
 - e) Type parameter names among different generic methods must be unique.
ANS: False. Type parameter names among different generic methods need not be unique.
 - f) The scope of a generic class's type parameter is the entire class.
ANS: True.
 - g) A type parameter can have at most one interface constraint, but multiple class constraints.
ANS: False. A type parameter can have at most one class constraint, but multiple interface constraints.
- 22.2** Fill in the blanks in each of the following:
- a) _____ enable you to specify, with a single method declaration, a set of related methods; _____ enable you to specify, with a single class declaration, a set of related classes.
ANS: Generic methods, Generic classes.
 - b) A type parameter list is delimited by _____.
ANS: angle brackets.
 - c) The _____ of a generic method can be used to specify the types of the arguments to the method, to specify the return type of the method and to declare variables within the method.
ANS: type parameters.
 - d) The statement "Stack< int > objectStack = new Stack< int >();" indicates that objectStack stores _____.
ANS: ints.
 - e) In a generic class declaration, the class name is followed by a(n) _____.
ANS: type parameter list.
 - f) The _____ constraint requires that the type argument must have a public parameterless constructor.
ANS: new.

Exercises

- 22.3** (*Generic Notation*) Explain the use of the following notation in a C# program:

```
public class Array<T>
```

ANS: This notation begins the declaration of a generic class named Array. The type parameter T would typically be used to represent the type stored in an Array object.

22.4 (*Overloading Generic Methods*) How can generic methods be overloaded?

ANS: Two or more generic methods can specify the same method name but different method parameters. A generic method can also be overloaded by another generic method that has the same method name and a different number of type parameters, or by a generic method that has both a different number of type parameters and a different number of method parameters. In addition, a generic method can be overloaded by a non-generic method that has the same method name and number of parameters, or by a non-generic method that has the same method name and a different number of method parameters.

22.5 (*Determining which Method to Call*) The compiler performs a matching process to determine which method to call when a method is invoked. Under what circumstances does an attempt to make a match result in a compile-time error?

ANS: If the compiler cannot match the method call made to either a non-generic method or a generic method, or if there is ambiguity due to multiple possible best matches, the compiler generates an error.

22.6 (*What Does this Statement Do?*) Explain why a C# program might use the statement

```
Array< Employee > workerlist = new Array< Employee >();
```

ANS: When creating objects from a generic class, it is necessary to provide a type argument (or possibly several type arguments) to instantiate the objects with actual types. The preceding statement would be used to create an Array object that presumably stores Employee objects. The compiler can then perform type checking to ensure that the code uses the Array of Employees properly.

22.7 (*Generic Linear Search Method*) Write a generic method, Search, that implements the linear-search algorithm. Method Search should compare the search key with each element in the array until the search key is found or until the end of the array is reached. If the search key is found, return its location in the array; otherwise, return -1. Write a test application that inputs and searches an int array and a double array. Provide buttons that the user can click to randomly generate int and double values. Display the generated values in a TextBox, so the user knows what values they can search for [*Hint: Use (T : IComparable< T >)* in the where clause for method Search so that you can use method CompareTo to compare the search key to the elements in the array.]

ANS:

```

1 // Exercise 22.7 Solution: GenericLinearSearch.cs
2 // Linear search of an array using a generic method
3 using System;
4
5 namespace GenericLinearSearch
6 {
7     class GenericLinearSearch
8     {
9         // iterate through array
10        public static int Search< T >( T key, T[] objects )
11            where T : IComparable< T >
12        {
13            // statement iterates linearly through an array
14            for ( int i = 0; i < objects.Length; i++ )
15            {
16                if ( objects[ i ].CompareTo( key ) == 0 )
17                    return i; // return the index of the key
18            } // end for

```

```

19
20         return -1; // indicates the key was not found
21     } // end method Search
22 } // end class GenericLinearSearch
23 } // end namespace GenericLinearSearch

```

```

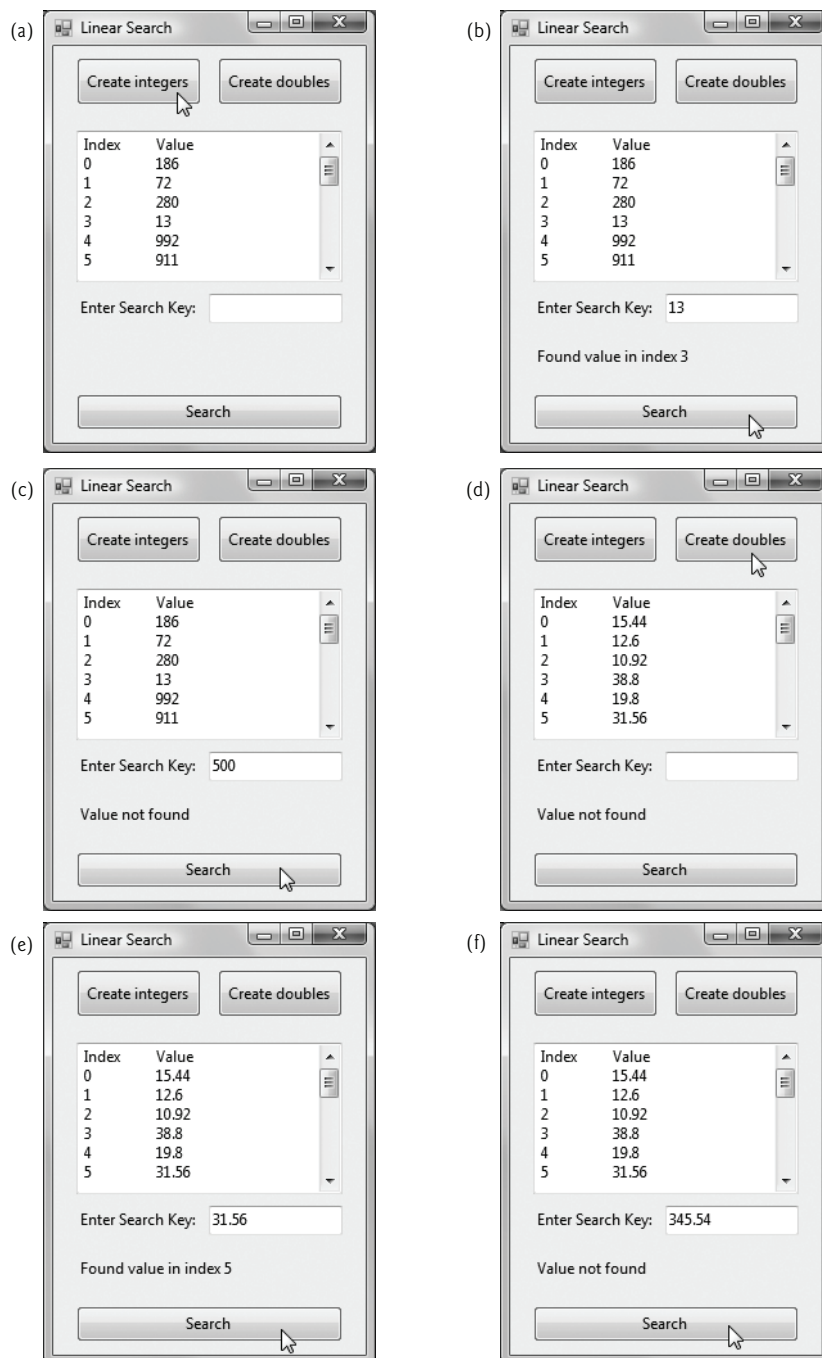
1 // Exercise 22.7 Solutions: GenericLinearSearchForm.cs
2 // Linear search of an array using a generic method.
3 using System;
4 using System.Windows.Forms;
5
6 namespace GenericLinearSearch
7 {
8     public partial class GenericLinearSearchForm : Form
9     {
10         int[] array1 = new int[ 20 ];
11         double[] array2 = new double[ 20 ];
12         bool createInt;
13
14         // constructor
15         public GenericLinearSearchForm()
16         {
17             InitializeComponent();
18         } // end GeneralLinearSearchTest constructor
19
20         // create random ints
21         private void createIntegersButton_Click(
22             object sender, EventArgs e )
23         {
24             Random randomNumber = new Random();
25             string output = "Index\tValue\r\n";
26
27             // create string containing 20 random numbers
28             for ( int i = 0; i < array1.Length; i++ )
29             {
30                 array1[ i ] = randomNumber.Next( 1000 );
31                 output += ( i + "\t" + array1[ i ] + "\r\n" );
32             } // end for
33
34             // specify whether to test the generic method with ints
35             // (true) or doubles (false)
36             createInt = true;
37             displayTextBox.Text = output; // display numbers
38             inputTextBox.Clear(); // clear search key text box
39             searchButton.Enabled = true; // enable search button
40         } // end method createIntegersButton_Click
41
42         // create random doubles
43         private void createDoublesButton_Click(
44             object sender, EventArgs e )
45         {
46             Random randomNumber = new Random();
47             string output = "Index\tValue\r\n";

```

```

48
49 // create string containing 20 random numbers
50 for ( int i = 0; i < array2.Length; i++ )
51 {
52     array2[ i ] = randomNumber.Next( 1000 ) / 25.0;
53     output += ( i + "\t" + array2[ i ] + "\r\n" );
54 } // end for
55
56 createInt = false; // indication for searching double
57 displayTextBox.Text = output; // display numbers
58 inputTextBox.Clear(); // clear search key text box
59 searchButton.Enabled = true; // enable search button
60 } // end method createDoublesButton_Click
61
62 // search array for search key
63 private void searchButton_Click( object sender, EventArgs e )
64 {
65     int result; // search result
66
67     // if search key text box is empty,
68     // display message and exit method
69     if ( string.IsNullOrEmpty( inputTextBox.Text ) )
70     {
71         MessageBox.Show( "You must enter a search key.", "Error",
72             MessageBoxButtons.OK, MessageBoxIcon.Error );
73         return;
74     } // end if
75
76     // search key is an int
77     if ( createInt )
78     {
79         int searchKey = Convert.ToInt32( inputTextBox.Text );
80         result = GenericLinearSearch.Search( searchKey, array1 );
81     } // end if
82     else
83     {
84         double searchKey = Convert.ToDouble( inputTextBox.Text );
85         result = GenericLinearSearch.Search( searchKey, array2 );
86     } // end else
87
88     // display search result
89     if ( result != -1 )
90         resultLabel.Text = "Found value in index " + result;
91     else
92         resultLabel.Text = "Value not found";
93 } // end method searchButton_Click
94 } // end class GenericLinearSearchForm
95 } // end namespace GenericLinearSearch

```



22.8 (Overloading a Generic Method) Overload generic method `DisplayArray` of Fig. 27.3 so that it takes two additional `int` arguments: `lowIndex` and `highIndex`. A call to this method displays only the designated portion of the array. Validate `lowIndex` and `highIndex`. If either is out of range, or if `highIndex` is less than or equal to `lowIndex`, the overloaded `DisplayArray` method should throw an `InvalidIndexException`; otherwise, `DisplayArray` should return the number of elements displayed. Then modify `Main` to exercise both versions of `DisplayArray` on arrays `intArray`, `doubleArray` and `charArray`. Test all capabilities of both versions of `DisplayArray`.

ANS:

```

1  // Exercise 22.8 Solution: InvalidIndexException.cs
2  // Indicates an invalid index.
3  using System;
4
5  class InvalidIndexException : ApplicationException
6  {
7      // parameterless constructor
8      public InvalidIndexException()
9          : base( "Invalid index." )
10     {
11         // empty constructor
12     } // end InvalidIndexException constructor
13
14     // constructor for customizing error message
15     public InvalidIndexException( string message )
16         : base( message )
17     {
18         // empty constructor
19     } // end InvalidIndexException constructor
20
21     // constructor for customizing error message and
22     // specifying InnerException object
23     public InvalidIndexException( string message, Exception inner )
24         : base( message, inner )
25     {
26         // empty constructor
27     } // end InvalidIndexException constructor
28 } // end class InvalidIndexException

```

```

1  // Exercise 22.8 Solution: GenericMethod.cs
2  // Using overloaded methods to display arrays of different types.
3  using System;
4
5  class GenericMethod
6  {
7      public static void Main( string[] args )
8      {
9          // create arrays of int, double and char
10         int[] intArray = { 1, 2, 3, 4, 5, 6 };
11         double[] doubleArray = { 1.1, 2.2, 3.3, 4.4, 5.5, 6.6, 7.7 };
12         char[] charArray = { 'H', 'E', 'L', 'L', 'O' };
13

```

```

14     Test( "intArray", intArray );
15     Test( "doubleArray", doubleArray );
16     Test( "charArray", charArray );
17 } // end Main
18
19 // generic method PrintArray
20 static void DisplayArray< T >( T[] inputArray )
21 {
22     // display array elements
23     foreach ( T element in inputArray )
24         Console.Write( "{0} ", element );
25
26     Console.WriteLine();
27 } // end method DisplayArray
28
29 // overload generic method DisplayArray
30 static int DisplayArray< T >( T[] inputArray, int lowIndex,
31     int highIndex )
32 {
33     // check if subscript is negative or out of range
34     if ( lowIndex < 0 || highIndex >= inputArray.Length )
35         throw new InvalidIndexException();
36
37     int count = 0;
38
39     // display array
40     for ( int i = lowIndex; i <= highIndex; i++ )
41     {
42         ++count;
43         Console.Write( "{0} ", inputArray[ i ] );
44     } // end for
45
46     Console.WriteLine();
47     return count;
48 } // end method DisplayArray
49
50 // generic method that tests both DisplayArray methods
51 static void Test< T >( string name, T[] array )
52 {
53     // display array
54     try
55     {
56         int elements; // store value returned by DisplayArray
57
58         // display an array using the original DisplayArray method
59         Console.WriteLine(
60             "\n\nUsing the original DisplayArray method" );
61         DisplayArray( array ); // display array
62
63         // display an array using the new DisplayArray method
64         Console.WriteLine( "Array {0} contains:", name );
65         elements = DisplayArray( array, 0, array.Length - 1 );
66         Console.WriteLine( "{0} elements were output", elements );
67

```

```

68         // display elements 1-3 of array
69         Console.WriteLine(
70             "Array {0} from positions 1 to 3 is", name );
71         elements = DisplayArray( array, 1, 3 );
72         Console.WriteLine( "{0} elements were output", elements );
73
74         // try to display an invalid element
75         Console.WriteLine( "Array {0} output with invalid indices",
76             name );
77         elements = DisplayArray( array, -1, 10 );
78     } // end try
79     catch ( InvalidIndexException exception )
80     {
81         Console.WriteLine( exception.StackTrace );
82     } // end catch
83 } // end method Test
84 } // end class GenericMethod

```

```

Using the original DisplayArray method
1 2 3 4 5 6
Array intArray contains:
1 2 3 4 5 6
6 elements were output
Array intArray from positions 1 to 3 is
2 3 4
3 elements were output
Array intArray output with invalid indices
    at GenericMethod.DisplayArray[T](T[] inputArray, Int32 lowIndex, Int32
    highIndex) in C:\solutions\sol_ch27\Ex27_08\GenericMethod.cs:line 35
    at GenericMethod.Test[T](String name, T[] array) in
    C:\solutions\sol_ch27\Ex27_08\GenericMethod.cs:line 77

```

```

Using the original DisplayArray method
1.1 2.2 3.3 4.4 5.5 6.6 7.7
Array doubleArray contains:
1.1 2.2 3.3 4.4 5.5 6.6 7.7
7 elements were output
Array doubleArray from positions 1 to 3 is
2.2 3.3 4.4
3 elements were output
Array doubleArray output with invalid indices
    at GenericMethod.DisplayArray[T](T[] inputArray, Int32 lowIndex, Int32
    highIndex) in C:\solutions\sol_ch27\Ex27_08\GenericMethod.cs:line 35
    at GenericMethod.Test[T](String name, T[] array) in
    C:\solutions\sol_ch27\Ex27_08\GenericMethod.cs:line 77

```

```

Using the original DisplayArray method
H E L L O
Array charArray contains:
H E L L O
5 elements were output
Array charArray from positions 1 to 3 is
E L L
3 elements were output
Array charArray output with invalid indices
  at GenericMethod.DisplayArray[T](T[] inputArray, Int32 lowIndex, Int32
    highIndex) in C:\solutions\sol_ch27\Ex27_08\GenericMethod.cs:line 35
  at GenericMethod.Test[T](String name, T[] array) in
    C:\solutions\sol_ch25\Ex25_08\GenericMethod.cs:line 77

```

22.9 (*Overloading a Generic Method with a Non-Generic Method*) Overload generic method `DisplayArray` of Fig. 27.3 with a nongeneric version that displays an array of strings in neat, tabular format, as shown in the sample output that follows:

```

Array stringArray contains:
one      two      three     four
five     six      seven     eight

```

ANS:

```

1  // Exercise 22.9 Solution: GenericMethod.cs
2  // Overloading generic methods with non-generic methods.
3  using System;
4
5  class GenericMethod
6  {
7      public static void Main( string[] args )
8      {
9          // create arrays of int, double and char
10         int[] intArray = { 1, 2, 3, 4, 5, 6 };
11         double[] doubleArray = { 1.1, 2.2, 3.3, 4.4, 5.5, 6.6, 7.7 };
12         char[] charArray = { 'H', 'E', 'L', 'L', 'O' };
13         string[] stringArray = { "one", "two", "three", "four",
14             "five", "six", "seven", "eight" };
15
16         Console.WriteLine( "Array intArray contains:" );
17         DisplayArray( intArray ); // pass an int array
18         Console.WriteLine( "Array doubleArray contains:" );
19         DisplayArray( doubleArray ); // pass a double array
20         Console.WriteLine( "Array charArray contains:" );
21         DisplayArray( charArray ); // pass a char array
22         Console.WriteLine( "Array stringArray contains:" );
23         DisplayArray( stringArray ); // method specific to string array
24     } // end Main
25
26     // generic method DisplayArray
27     static void DisplayArray< T >( T[] inputArray )
28     {

```

```

29     // display array elements
30     foreach ( T element in inputArray )
31         Console.Write( "{0} ", element );
32
33     Console.WriteLine( "\n" );
34 } // end method DisplayArray
35
36 // method that displays an array of strings in tabular format
37 static void DisplayArray( string[] stringArray )
38 {
39     // display elements of array
40     for ( int i = 0; i < stringArray.Length; i++ )
41     {
42         Console.Write( "{0, -10}", stringArray[ i ] );
43
44         // create rows
45         if ( ( i + 1 ) % 4 == 0 )
46             Console.WriteLine();
47     } // end for
48 } // end method DisplayArray
49 } // end class GenericMethod

```

Array intArray contains:

1 2 3 4 5 6

Array doubleArray contains:

1.1 2.2 3.3 4.4 5.5 6.6 7.7

Array charArray contains:

H E L L O

Array stringArray contains:

one	two	three	four
five	six	seven	eight

22.10 (Generic Method *IsEqualTo*) Write a simple generic version of method *IsEqualTo* that compares its two arguments with the *Equals* method, and returns *true* if they're equal and *false* otherwise. Use this generic method in a program that calls *IsEqualTo* with a variety of simple types, such as *object* or *int*. What result do you get when you attempt to run this program?

ANS: For classes that override the *Equals* method to compare the contents of the objects, the program will compare the objects based on their contents. For classes that do not override the *Equals* method, the program will compare the objects based on their references, not their contents. So, even though they look the same when printed in string format (see last line of sample output), two separate objects are not considered equal when method *Equals* is not overloaded for their class.

```

1 // Exercise 22.10 Solution: MethodIsEqualToTest.cs
2 // Testing generic method IsEqualTo.
3 using System;
4
5 class MethodIsEqualToTest
6 {

```

```
7 public static void Main( string[] args )
8 {
9     string result; // store a string based on what IsEqualTo returns
10    int a; // int used for testing equality
11    int b; // int used for testing equality
12
13    // get two ints from the user
14    Console.Write( "Enter first integer: " );
15    a = Convert.ToInt32( Console.ReadLine() );
16    Console.Write( "Enter second integer: " );
17    b = Convert.ToInt32( Console.ReadLine() );
18
19    // test if the two ints input by user are equal
20    if ( IsEqualTo( a, b ) )
21        result = "equal";
22    else
23        result = "not equal";
24
25    Console.WriteLine( "{0} and {1} are {2}\n", a, b, result );
26
27    string c; // string used for testing equality
28    string d; // string used for testing equality
29
30    // get two strings from the user
31    Console.Write( "Enter first string: " );
32    c = Console.ReadLine();
33    Console.Write( "Enter second string: " );
34    d = Console.ReadLine();
35
36    // test if two strings input by user are equal
37    if ( IsEqualTo( c, d ) )
38        result = "equal";
39    else
40        result = "not equal";
41
42    Console.WriteLine( "{0} and {1} are {2}\n", c, d, result );
43
44    double e; // double used for testing equality
45    double f; // double used for testing equality
46
47    // get two doubles from user
48    Console.Write( "Enter first double: " );
49    e = Convert.ToDouble( Console.ReadLine() );
50    Console.Write( "Enter second double: " );
51    f = Convert.ToDouble( Console.ReadLine() );
52
53    // test if two doubles input by user are equal
54    if ( IsEqualTo( e, f ) )
55        result = "equal";
56    else
57        result = "not equal";
58
59    Console.WriteLine( "{0} and {1} are {2}\n", e, f, result );
60
```

```

61     object g = new object(); // object used for testing equality
62     object h = new object(); // object used for testing equality
63
64     // test if two objects are equal
65     if ( IsEqualTo( g, h ) )
66         result = "equal";
67     else
68         result = "not equal";
69
70     Console.WriteLine( "{0} and {1} are {2}", g, h, result );
71 } // end Main
72
73 // test whether two generic types are equal
74 static bool IsEqualTo< T >( T first, T second )
75 {
76     return first.Equals( second );
77 } // end method IsEqualTo
78 } // end class MethodIsEqualToTest

```

```

Enter first int: 12
Enter second int: 8
12 and 8 are not equal

```

```

Enter first string: test
Enter second string: test
test and test are equal

```

```

Enter first double: 1.2
Enter second double: 2.1
1.2 and 2.1 are not equal

```

```

System.Object and System.Object are not equal

```

22.11 (Generic Class Pair) Write a generic class *Pair* which has two type parameters, *F* and *S*, representing the type of the first and second element of the pair, respectively. Add properties for the first and second elements of the pair. [*Hint:* The class header should be `public class Pair<F, S>.`]

ANS:

```

1  // Exercise 22.11 Solution: Pair.cs
2  // Pair contains two types
3  using System;
4
5  public class Pair< F, S >
6  {
7      // constructor
8      public Pair( F firstElement, S secondElement )
9      {
10         First = firstElement;
11         Second = secondElement;
12     } // end Pair constructor
13

```

```

14 // auto-implemented property First
15 public F First { get; set; }
16
17 // auto-implemented property Second
18 public S Second { get; set; }
19 } // end class Pair

```

```

1 // Exercise 22.11 Solution: PairTest.cs
2 // Generic Pair class testing program.
3 using System;
4
5 class PairTest
6 {
7     public static void Main( string[] args )
8     {
9         // create a pair of int and string
10        Pair< int, string > numberPair =
11            new Pair< int, string >( 1, "one" );
12
13        // display original numberPair
14        Console.WriteLine( "Original pair: < {0}, {1} >",
15            numberPair.First, numberPair.Second );
16
17        // modify pair
18        numberPair.First = 2;
19        numberPair.Second = "two";
20
21        // display modified numberPair
22        Console.WriteLine( "Modified pair: < {0}, {1} >",
23            numberPair.First, numberPair.Second );
24    } // end Main
25 } // end class PairTest

```

```

Original pair: < 1, one >
Modified pair: < 2, two >

```

22.12 (Generic Classes *TreeNode* and *Tree*) Convert classes *TreeNode* and *Tree* from Figs. 26.19 and 26.20 into generic classes. To insert an object in a *Tree*, the object must be compared to the objects in existing *TreeNode*s. For this reason, classes *TreeNode* and *Tree* should specify *IComparable<T>* as the interface constraint of each class's type parameter. After modifying classes *TreeNode* and *Tree*, write a test application that creates three *Tree* objects—one that stores ints, one that stores doubles and one that stores strings. Insert 10 values into each tree. Then output the preorder, inorder and postorder traversals for each *Tree*.

ANS:

```

1 // Exercise 22.12 Solution: TreeNode.cs
2 // Generic class TreeNode represents a node in a Tree.
3 using System;
4

```

```

5 public class TreeNode< T > where T : IComparable< T >
6 {
7     // initialize data and make that a leaf node
8     public TreeNode( T nodeData )
9     {
10         Data = nodeData; // set value of node
11         RightNode = null; // node has no children
12         LeftNode = null; // node has no children
13     } // end TreeNode constructor
14
15     // auto-implemented property LeftNode
16     public TreeNode< T > LeftNode { get; set; }
17
18     // auto-implemented property Data
19     public T Data { get; set; }
20
21     // auto-implemented property RightNode
22     public TreeNode< T > RightNode { get; set; }
23
24     // insert node into tree
25     public void Insert( T insertValue )
26     {
27         // insert in the left subtree
28         if ( insertValue.CompareTo( Data ) < 0 )
29             // insert a new TreeNode
30             if ( LeftNode == null )
31                 LeftNode = new TreeNode< T >( insertValue );
32             else // continue traversing the left subtree
33                 LeftNode.Insert( insertValue );
34
35         // insert in the right subtree
36         else if ( insertValue.CompareTo( Data ) > 0 )
37         {
38             // insert a new TreeNode
39             if ( RightNode == null )
40                 RightNode = new TreeNode< T >( insertValue );
41             else // continue traversing the right subtree
42                 RightNode.Insert( insertValue );
43         } // end else if
44     } // end method Insert
45 } // end class TreeNode

```

```

1 // Exercise 22.12 Solution: Tree.cs
2 // Generic class Tree is a tree containing TreeNodes.
3 using System;
4
5 public class Tree< T > where T : IComparable< T >
6 {
7     private TreeNode< T > root;
8
9     // construct an empty Tree
10    public Tree()
11    {

```

```
12     root = null;
13 } // end Tree constructor
14
15 // insert a new node in a binary search tree
16 public void InsertNode( T insertValue )
17 {
18     // if node does not exist, create node
19     if ( root == null )
20         root = new TreeNode< T >( insertValue );
21     else // otherwise insert node into tree
22         root.Insert( insertValue );
23 } // end method InsertNode
24
25 // begin the preorder traversal
26 public void PreorderTraversal()
27 {
28     PreorderHelper( root );
29 } // end method PreOrderTraversal
30
31 // recursive method to perform preorder traversal
32 private void PreorderHelper( TreeNode< T > node )
33 {
34     if ( node == null )
35         return;
36
37     Console.Write( node.Data.ToString() + " " );
38     // output the node data
39     PreorderHelper( node.LeftNode ); // traverse the left subtree
40     PreorderHelper( node.RightNode ); // traverse the right subtree
41 } // end method PreorderHelper
42
43 // begin the inorder traversal
44 public void InorderTraversal()
45 {
46     InorderHelper( root );
47 } // end method InorderTraversal
48
49 // recursive method to perform inorder traversal
50 private void InorderHelper( TreeNode< T > node )
51 {
52     if ( node == null )
53         return;
54
55     InorderHelper( node.LeftNode ); // traverse the left subtree
56     Console.Write( node.Data.ToString() + " " );
57     // output the node data
58     InorderHelper( node.RightNode ); // traverse the right subtree
59 } // end method InorderHelper
60
61 // begin the postorder traversal
62 public void PostorderTraversal()
63 {
64     PostorderHelper( root );
65 } // end method PostorderTraversal
```

```

66
67 // recursive method to perform postorder traversal
68 private void PostorderHelper( TreeNode< T > node )
69 {
70     if ( node == null )
71         return;
72
73     PostorderHelper( node.LeftNode ); // traverse the left subtree
74     PostorderHelper( node.RightNode ); // traverse the right subtree
75     Console.Write( node.Data.ToString() + " " );
76     // output the node data
77 } // end method PostorderHelper
78 } // end class Tree

```

```

1 // Exercise 22.12 Solution: GenericTreeTest.cs
2 // This program tests generic class Tree.
3 using System;
4
5 class GenericTreeTest
6 {
7     public static void Main( string[] args )
8     {
9         // set up and manipulate a Tree< int >
10        Tree< int > intTree = new Tree< int >();
11        int[] intArray =
12            new int[] { 47, 25, 77, 11, 43, 65, 93, 7, 17, 68 };
13
14        Console.WriteLine(
15            "Inserting the following values in the intTree:" );
16
17        // insert elements of intArray in intTree
18        foreach ( var element in intArray )
19        {
20            Console.Write( "{0} ", element );
21            intTree.InsertNode( element );
22        } // end foreach
23
24        Console.WriteLine( "\n\nPreorder traversal of intTree" );
25        intTree.PreorderTraversal(); // perform preorder traversal
26
27        Console.WriteLine( "\n\nInorder traversal of intTree" );
28        intTree.InorderTraversal(); // perform inorder traversal
29
30        Console.WriteLine( "\n\nPostorder traversal of intTree" );
31        intTree.PostorderTraversal(); // perform postorder traversal
32
33        // set up and manipulate Tree< double >
34        Tree< double > doubleTree = new Tree< double >();
35        double[] doubleArray = new double[] {
36            4.7, 2.5, 7.7, 1.1, 4.3, 6.5, 9.3, 0.7, 1.7, 6.8 };
37        Console.WriteLine(
38            "\n\nInserting the following values in the doubleTree" );
39

```

```

40      // insert the elements of doubleArray in doubleTree
41      foreach ( var element in doubleArray )
42      {
43          Console.Write( "{0} ", element );
44          doubleTree.InsertNode( element );
45      } // end foreach
46
47      Console.WriteLine( "\n\nPreorder traversal of doubleTree" );
48      doubleTree.PreorderTraversal(); // perform preorder traversal
49
50      Console.WriteLine( "\n\nInorder traversal of doubleTree" );
51      doubleTree.InorderTraversal(); // perform inorder traversal
52
53      Console.WriteLine( "\n\nPostorder traversal of doubleTree" );
54      doubleTree.PostorderTraversal(); // perform postorder traversal
55
56      // set up and manipulate Tree< string >
57      Tree< string > stringTree = new Tree< string >();
58      string[] stringArray = new string[] {
59          "green", "blue", "red", "black", "cyan", "orange",
60          "white", "yellow", "magenta", "gray" };
61
62      Console.WriteLine(
63          "\n\nInserting the following values in the stringTree" );
64
65      // insert elements of stringArray in stringTree
66      foreach ( var element in stringArray )
67      {
68          Console.Write( "{0} ", element );
69          stringTree.InsertNode( element );
70      } // end foreach
71
72      Console.WriteLine( "\n\nPreorder traversal of stringTree" );
73      stringTree.PreorderTraversal(); // perform preorder traversal
74
75      Console.WriteLine( "\n\nInorder traversal of stringTree" );
76      stringTree.InorderTraversal(); // perform inorder traversal
77
78      Console.WriteLine( "\n\nPostorder traversal of stringTree" );
79      stringTree.PostorderTraversal(); // perform postorder traversal
80      Console.WriteLine();
81  } // end Main
82 } // end class GenericTreeTest

```

Inserting the following values in the intTree:

47 25 77 11 43 65 93 7 17 68

Preorder traversal of intTree

47 25 11 7 17 43 77 65 68 93

Inorder traversal of intTree

7 11 17 25 43 47 65 68 77 93

Postorder traversal of intTree

7 17 11 43 25 68 65 93 77 47

Inserting the following values in the doubleTree
 4.7 2.5 7.7 1.1 4.3 6.5 9.3 0.7 1.7 6.8

Preorder traversal of doubleTree
 4.7 2.5 1.1 0.7 1.7 4.3 7.7 6.5 6.8 9.3

Inorder traversal of doubleTree
 0.7 1.1 1.7 2.5 4.3 4.7 6.5 6.8 7.7 9.3

Postorder traversal of doubleTree
 0.7 1.7 1.1 4.3 2.5 6.8 6.5 9.3 7.7 4.7

Inserting the following values in the stringTree
 green blue red black cyan orange white yellow magenta gray

Preorder traversal of stringTree
 green blue black cyan gray red orange magenta white yellow

Inorder traversal of stringTree
 black blue cyan gray green magenta orange red white yellow

Postorder traversal of stringTree
 black gray cyan blue magenta orange yellow white red green

22.13 (Generic Method TestTree) Modify your test program from Exercise 22.12 to use generic method TestTree to test the three Tree objects. The method should be called three times—once for each Tree object.

ANS:

```

1  // Exercise 22.13 Solution: GenericTreeTest.cs
2  // Testing generic class Tree with generic method TestTree.
3  using System;
4
5  class GenericTreeTest
6  {
7      public static void Main( string[] args )
8      {
9          // set up and manipulate Tree< int >
10         Tree< int > intTree = new Tree< int >();
11         int[] intArray =
12             new int[] { 47, 25, 77, 11, 43, 65, 93, 7, 17, 68 };
13
14         TestTree( "intTree", intArray, intTree );
15
16         // set up and manipulate Tree< double >
17         Tree< double > doubleTree = new Tree< double >();
18         double[] doubleArray = new double[] {
19             4.7, 2.5, 7.7, 1.1, 4.3, 6.5, 9.3, 0.7, 1.7, 6.8 };
20
21         TestTree( "doubleTree", doubleArray, doubleTree );
22

```

```

23 // set up and manipulate Tree< string >
24 Tree< string > stringTree = new Tree< string >();
25 string[] stringArray = new string[] {
26     "green", "blue", "red", "black", "cyan", "orange",
27     "white", "yellow", "magenta", "gray" };
28
29 TestTree( "stringTree", stringArray, stringTree );
30 } // end Main
31
32 // generic method that tests generic class Tree
33 static void TestTree< T >( string treeName,
34     T[] initializers, Tree< T > tree ) where T : IComparable< T >
35 {
36     Console.WriteLine(
37         "\n\nInserting the following values in the {0}", treeName );
38
39     // insert elements in initializers into Tree
40     foreach ( T element in initializers )
41     {
42         Console.Write( "{0} ", element );
43         tree.InsertNode( element );
44     } // end foreach
45
46     Console.WriteLine(
47         "\n\nPreorder traversal of {0}", treeName );
48     tree.PreorderTraversal(); // perform preorder traversal
49
50     Console.WriteLine(
51         "\n\nInorder traversal of {0}", treeName );
52     tree.InorderTraversal(); // perform inorder traversal
53
54     Console.WriteLine(
55         "\n\nPostorder traversal of {0}", treeName );
56     tree.PostorderTraversal(); // perform postorder traversal
57     Console.WriteLine();
58 } // end method TestTree
59 } // end class GenericTreeTest

```

Inserting the following values in the intTree
47 25 77 11 43 65 93 7 17 68

Preorder traversal of intTree
47 25 11 7 17 43 77 65 68 93

Inorder traversal of intTree
7 11 17 25 43 47 65 68 77 93

Postorder traversal of intTree
7 17 11 43 25 68 65 93 77 47

Inserting the following values in the doubleTree
4.7 2.5 7.7 1.1 4.3 6.5 9.3 0.7 1.7 6.8

Preorder traversal of doubleTree
4.7 2.5 1.1 0.7 1.7 4.3 7.7 6.5 6.8 9.3

Inorder traversal of doubleTree
0.7 1.1 1.7 2.5 4.3 4.7 6.5 6.8 7.7 9.3

Postorder traversal of doubleTree
0.7 1.7 1.1 4.3 2.5 6.8 6.5 9.3 7.7 4.7

Inserting the following values in the stringTree
green blue red black cyan orange white yellow magenta gray

Preorder traversal of stringTree
green blue black cyan gray red orange magenta white yellow

Inorder traversal of stringTree
black blue cyan gray green magenta orange red white yellow

Postorder traversal of stringTree
black gray cyan blue magenta orange yellow white red green

23

Collections: Solutions

*The shapes a bright container
can contain!*

—Theodore Roethke

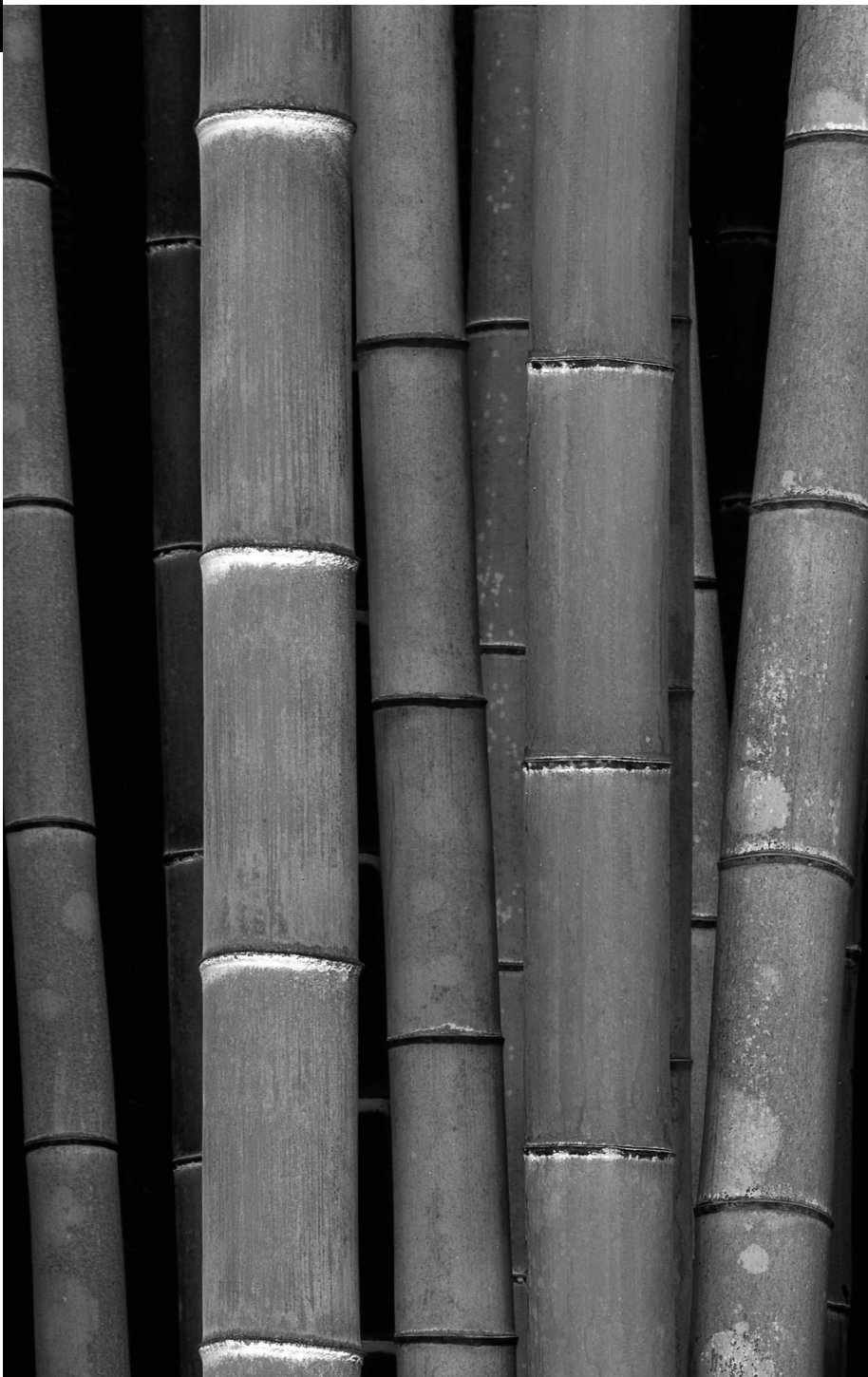
*I think this is the most
extraordinary collection of
talent, of human knowledge,
that has ever been gathered
together at the White House—
with the possible exception of
when Thomas Jefferson dined
alone.*

—John F. Kennedy

Objectives

In this chapter you'll learn:

- The nongeneric and generic collections that are provided by the .NET Framework.
- To use class `Array`'s `static` methods to manipulate arrays.
- To use enumerators to “walk through” a collection.
- To use the `foreach` statement with the .NET collections.
- To use nongeneric collection classes `ArrayList`, `Stack`, and `Hashtable`.
- To use generic collection classes `SortedDictionary` and `LinkedList`.



Self-Review Exercises

23.1 Fill in the blanks in each of the following statements:

- a) A(n) _____ is used to walk through a collection but cannot remove elements from the collection during the iteration.

ANS: enumerator (or foreach statement).

- b) Class _____ provides the capabilities of an arraylike data structure that can resize itself dynamically.

ANS: ArrayList.

- c) An element in an ArrayList can be accessed by using the ArrayList's _____.

ANS: indexer.

- d) IEnumerator method _____ advances the enumerator to the next item.

ANS: MoveNext.

- e) If the collection it references has been altered since the enumerator's creation, calling method Reset will cause an _____.

ANS: InvalidOperationException.

23.2 State whether each of the following is *true* or *false*. If *false*, explain why.

- a) Class Stack is in the System.Collections namespace.

ANS: True.

- b) A hashtable stores key/value pairs.

ANS: True.

- c) A class implementing interface IEnumerator must define only methods MoveNext and Reset, and no properties.

ANS: False. The class must also implement property Current.

- d) Values of simple types may be stored directly in an ArrayList.

ANS: False. An ArrayList stores only objects. Autoboxing occurs when adding a value type to the ArrayList. You can prevent boxing by instead using generic class List with a value type.

- e) An ArrayList can contain duplicate values.

ANS: True.

- f) A Hashtable can contain duplicate keys.

ANS: False. A Hashtable cannot contain duplicate keys.

- g) A LinkedList can contain duplicate values.

ANS: True.

- h) Dictionary is an interface.

ANS: False. Dictionary is a class; IDictionary is an interface.

- i) Enumerators can change the values of elements but cannot remove them.

ANS: False. An enumerator cannot be used to change the values of elements.

- j) With hashing, as the load factor increases, the chance of collisions decreases.

ANS: False. With hashing, as the load factor increases, there are fewer available slots relative to the total number of slots, so the chance of selecting an occupied slot (a collision) with a hashing operation increases.

Exercises

23.3 (*Collections Terminology*) Define each of the following terms:

- a) ICollection

ANS: Interface ICollection is the root interface in the collection hierarchy from which interfaces IList and IDictionary are derived.

b) Array

ANS: Class Array provides static methods that manipulate arrays, and is the abstract base class of all array types.

c) IList

ANS: An interface that describes the implementation for lists. It declares methods for manipulating lists, as well as an indexer for int-based access.

d) load factor

ANS: The ratio of the number of stored objects in a hash table to the size of the hash table.

e) Hashtable collision

ANS: A situation where two keys in a Hashtable hash into the same cell.

f) space/time trade-off in hashing

ANS: When the load factor is increased, the result is better memory utilization. However, the application runs slower due to increased hashing collisions.

g) Hashtable

ANS: A .NET Framework class that enables programmers to use hashing.

23.4 (ArrayList Methods) Explain briefly the operation of each of the following methods of class ArrayList:

a) Add

ANS: Appends the specified element to the ArrayList.

b) Insert

ANS: Inserts one element at the specified position.

c) Remove

ANS: Removes the first occurrence of an element from the ArrayList, and shifts all subsequent elements towards the beginning of the ArrayList to fill the emptied position.

d) Clear

ANS: Removes all elements from the ArrayList.

e) RemoveAt

ANS: Removes the element at the specified index, and shifts all subsequent elements towards the beginning of the ArrayList to fill the emptied position.

f) Contains

ANS: Determines whether an ArrayList contains a specified object.

g) IndexOf

ANS: Returns the index of the first occurrence of a specified object, or -1 if the object is not found.

h) Count

ANS: The current number of elements in the ArrayList.

i) Capacity

ANS: The number of elements available for storage (used and unused).

23.5 (ArrayList Insertion Performance) Explain why inserting additional elements into an ArrayList object whose current size is less than its capacity is a relatively fast operation and why inserting additional elements into an ArrayList object whose current size is at capacity is relatively slow.

ANS: An ArrayList whose current Count is less than its Capacity has memory available. Insertions are fast because new memory does not need to be allocated. An ArrayList that is at its capacity must have its memory reallocated and the existing values copied into it.

23.6 (Inheritance Negatives) In our implementation of a stack in Fig. 26.13, we were able to quickly extend a linked list to create class StackInheritance. The .NET Framework designers chose

not to use inheritance to create their `Stack` class. What are the negative aspects of inheritance, particularly for class `Stack`?

ANS: Operations can be performed on `Stack` objects that are not normally allowed, which can lead to corruption of the stack. For example, `List` method `InsertAtBack` in Fig. 26.4 should not be performed on a `Stack`.

23.7 (*Collections Short Questions*) Briefly answer the following questions:

a) What happens when you add a simple type value to a non-generic collection?

ANS: Autoboxing occurs when a simple type value is added to a nongeneric collection. The simple type value is packaged inside an object.

b) Can you display all the elements in an `IEnumerable` object without explicitly using an enumerator? If yes, how?

ANS: Yes, the elements in an `IEnumerable` object can be displayed by iterating through the collection with the `foreach` statement.

23.8 (*Enumerator Methods*) Explain briefly the operation of each of the following enumerator-related methods:

a) `GetEnumerator`

ANS: Returns an enumerator for a collection.

b) `Current`

ANS: Obtains the current element of the enumerator.

c) `MoveNext`

ANS: Returns `true` if the enumerator was successfully advanced to the next element.

23.9 (*HashTable Methods and Properties*) Explain briefly the operation of each of the following methods and properties of class `HashTable`:

a) `Add`

ANS: Adds a key/value pair into the `HashTable`.

b) `Keys`

ANS: Gets an `ICollection` of the keys of the `HashTable`.

c) `Values`

ANS: Gets an `ICollection` of the values stored in the `HashTable`.

d) `ContainsKey`

ANS: Determines whether the specified key is in the `HashTable`.

23.10 (*True/False*) Determine whether each of the following statements is *true* or *false*. If *false*, explain why.

a) Elements in an array must be sorted in ascending order before a `BinarySearch` may be performed.

ANS: `True`.

b) Method `First` gets the first node in a `LinkedList`.

ANS: `True`.

c) Class `Array` provides static method `Sort` for sorting array elements.

ANS: `True`.

23.11 (*LinkedList without Dups*) Write an application that reads in a series of first names and stores them in a `LinkedList`. Do not store duplicate names. Allow the user to search for a first name.

ANS:

```
1 // Exercise 23.11 Solution: ListTest.cs
2 // Application stores first names in LinkedList.
3 using System;
4 using System.Collections.Generic;
```

```
5
6 public class ListTest
7 {
8     private LinkedList< string > nameList;
9
10    // parameterless constructor
11    public ListTest()
12    {
13        nameList = new LinkedList< string >();
14
15        GetNames(); // get input from user
16        SearchNames(); // search for names
17    } // end constructor ListTest
18
19    // get names
20    public void GetNames()
21    {
22        // get name from standard input
23        Console.WriteLine( "{0}\n{1}",
24            "Add a name to list.",
25            "Type <Ctrl> z and press Enter to terminate input:" );
26        string inputName = Console.ReadLine();
27
28        // obtain input until end of file entered
29        while ( inputName != null )
30        {
31            // insert name
32            if ( !FoundName( inputName ) )
33            {
34                InsertName( inputName );
35                Console.WriteLine( inputName + " inserted" );
36            } // end if
37            else // name already exists in list
38                Console.WriteLine( inputName + " exists in list" );
39
40            // get next name
41            Console.WriteLine( "{0}\n{1}",
42                "Add a name to list.",
43                "Type <Ctrl> z and press Enter to terminate input:" );
44            inputName = Console.ReadLine();
45        } // end while
46    } // end method GetNames
47
48    // search names from list
49    public void SearchNames()
50    {
51        // get name from standard input
52        Console.WriteLine( "{0}\n{1}",
53            "Search a name.",
54            "Type <Ctrl> z and press Enter to terminate searching:" );
55        string inputName = Console.ReadLine();
56
57        // obtain input until end of file entered
58        while ( inputName != null )
59        {
```

```

60         // name found
61         if ( FoundName( inputName ) )
62             Console.WriteLine( inputName + " found in list" );
63         else // name not found
64             Console.WriteLine( inputName + " not found in list" );
65
66         // get next search name
67         Console.WriteLine( "{0}\n{1}",
68             "Search a name.",
69             "Type <Ctrl> z and press Enter to terminate searching:" );
70         inputName = Console.ReadLine();
71     } // end while
72 } // end method SearchNames
73
74 // search for name
75 public bool FoundName( string searchName )
76 {
77     // search the entire list with Find method
78     // (we also could have just used the Contains method)
79     return ( nameList.Find( searchName ) != null );
80 } // end method FoundName
81
82 // insert name
83 public void InsertName( string name )
84 {
85     nameList.AddLast( name );
86 } // end method InsertName
87
88 public static void Main( string[] args )
89 {
90     new ListTest();
91 } // end Main
92 } // end class ListTest

```

```

Add a name to list.
Type <Ctrl> z and press Enter to terminate input:
Moe
Moe inserted
Add a name to list.
Type <Ctrl> z and press Enter to terminate input:
Larry
Larry inserted
Add a name to list.
Type <Ctrl> z and press Enter to terminate input:
Curly
Curly inserted
Add a name to list.
Type <Ctrl> z and press Enter to terminate input:
Moe
Moe exists in list
Add a name to list.
Type <Ctrl> z and press Enter to terminate input:
^Z

```

```

Search a name.
Type <Ctrl> z and press Enter to terminate searching:
Harry
Harry not found in list
Search a name.
Type <Ctrl> z and press Enter to terminate searching:
Moe
Moe found in list
Search a name.
Type <Ctrl> z and press Enter to terminate searching:
^Z

```

23.12 (*Generic SortedDictionary*) Modify the application in Fig. 28.8 to count the number of occurrences of each letter rather than of each word. For example, the string "HELLO THERE" contains two Hs, three Es, two Ls, one O, one T and one R. Display the results.

ANS:

```

1  // Exercise 23.12 Solution: LetterCount.cs
2  // Application counts the number of occurrences
3  // of each letter in a string.
4  using System;
5  using System.Collections.Generic;
6
7  public class LetterCount
8  {
9      public static void Main( string[] args )
10     {
11         // create sorted dictionary based on user input
12         SortedDictionary< char, int > dictionary = CollectLetters();
13
14         // display sorted dictionary content
15         DisplayDictionary( dictionary );
16     } // end Main
17
18     // create sorted dictionary from user input
19     private static SortedDictionary< char, int > CollectLetters()
20     {
21         // create a new sorted dictionary
22         SortedDictionary< char, int > dictionary =
23             new SortedDictionary< char, int >();
24
25         Console.WriteLine( "Enter a string: " ); // prompt for user input
26         string input = Console.ReadLine(); // get input
27         input = input.ToLower(); // make string lowercase
28
29         // processing each character of the input
30         foreach ( char charKey in input )
31         {
32             // ensure that the character is a letter
33             if ( char.IsLetter( charKey ) )
34             {

```

```

35         // if the dictionary contains the letter
36         if ( dictionary.ContainsKey( charKey ) )
37         {
38             ++dictionary[ charKey ];
39         } // end inner if
40         else
41             // add new letter with a count of 1 to the dictionary
42             dictionary.Add( charKey, 1 );
43     } // end outer if
44 } // end foreach
45
46     return dictionary;
47 } // end method CollectLetters
48
49 // display dictionary content
50 private static void DisplayDictionary< K, V >(
51     SortedDictionary< K, V > dictionary )
52 {
53     Console.WriteLine( "\nSorted dictionary contains:\n{0,-12}{1,-12}",
54         "Key:", "Value:" );
55
56     // generate output for each key in the sorted dictionary
57     // by iterating through the Keys property with a foreach statement
58     foreach ( K key in dictionary.Keys )
59         Console.WriteLine( "{0,-12}{1,-12}", key, dictionary[ key ] );
60
61     Console.WriteLine( "\nsize: {0}", dictionary.Count );
62 } // end method DisplayDictionary
63 } // end class LetterCount

```

Enter a string:
To be or not to be: that is the question

Sorted dictionary contains:

Key:	Value:
a	1
b	2
e	4
h	2
i	2
n	2
o	5
q	1
r	1
s	2
t	7
u	1

size: 12

23.13 (*SortedDictionary of Colors*) Use a `SortedDictionary` to create a reusable class for choosing from some of the predefined colors in class `Color` (in the `System.Drawing` namespace). The names of the colors should be used as keys, and the predefined `Color` objects should be used as values. Place this class in a class library that can be referenced from any C# application. Use your new

class in a Windows application that allows the user to select a color, then changes the background color of the Form.

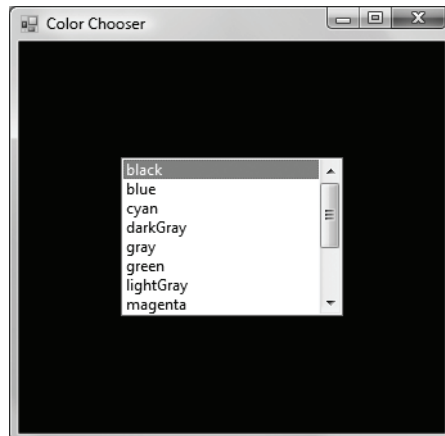
ANS:

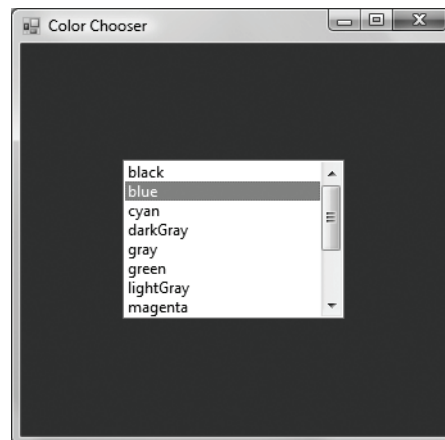
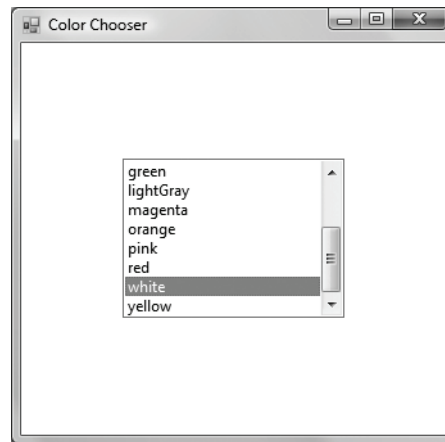
```

1  // Exercise 28.13 Solution: ColorChooser.cs
2  // Class that uses a SortedDictionary to store color name-color pairs.
3  using System;
4  using System.Collections.Generic;
5  using System.Drawing;
6
7  namespace Colors
8  {
9      public class ColorChooser
10     {
11         private SortedDictionary< string, Color > colorTable;
12
13         public ColorChooser()
14         {
15             colorTable = new SortedDictionary< string, Color >();
16
17             // add some colors to the colorTable
18             colorTable.Add( "black", Color.Black );
19             colorTable.Add( "blue", Color.Blue );
20             colorTable.Add( "cyan", Color.Cyan );
21             colorTable.Add( "darkGray", Color.DarkGray );
22             colorTable.Add( "gray", Color.Gray );
23             colorTable.Add( "green", Color.Green );
24             colorTable.Add( "lightGray", Color.LightGray );
25             colorTable.Add( "magenta", Color.Magenta );
26             colorTable.Add( "orange", Color.Orange );
27             colorTable.Add( "pink", Color.Pink );
28             colorTable.Add( "red", Color.Red );
29             colorTable.Add( "white", Color.White );
30             colorTable.Add( "yellow", Color.Yellow );
31         } // end constructor ColorChooser
32
33         // return the selected color
34         public Color GetColor( string name )
35         {
36             return colorTable[ name ];
37         } // end method GetColor
38
39         // return all the color names
40         public IEnumerable< string > GetKeys()
41         {
42             return colorTable.Keys;
43         } // end method GetKeys
44     } // end class ColorChooser
45 } // end namespace ColorChooser

```

```
1 // Exercise 27.13 Solution: ColorTestForm.cs
2 // A GUI to test the ColorChooser class.
3 using System;
4 using System.Windows.Forms;
5 using Colors;
6
7 namespace ColorChooserTest
8 {
9     public partial class ColorChooserForm : Form
10     {
11         private ColorChooser colors;
12
13         // constructor
14         public ColorChooserForm()
15         {
16             InitializeComponent();
17
18             // create a new color chooser
19             colors = new ColorChooser();
20
21             // fill the list box with the keys of colors
22             foreach ( var colorName in colors.GetKeys() )
23                 colorsListBox.Items.Add( colorName );
24         } // end constructor
25
26         // event handler that changes the background color based on user
27         // selection
28         private void colorsListBox_SelectedIndexChanged( object sender,
29             EventArgs e )
30         {
31             ListBox list = ( ListBox ) sender;
32
33             // set the background color to the selected color
34             BackColor = colors.GetColor( ( string ) list.SelectedItem );
35         } // end event handler
36     } // end class ColorChooserForm
37 } // end namespace ColorChooser
```





23.14 (*Duplicate Words in a Sentence*) Write an application that determines and displays the number of duplicate words in a sentence. Treat uppercase and lowercase letters the same. Ignore punctuation.

ANS:

```

1  // Exercise 23.14 Solution: Duplicates.cs
2  // Application stores the number of duplicate words in a sentence.
3  using System;
4  using System.Collections.Generic;
5  using System.Text.RegularExpressions;
6
7  public class Duplicates
8  {
9      public Duplicates()
10     {
11         Console.WriteLine( "Please enter a sentence:" );
12         string sentence = Console.ReadLine(); // read sentence
13
14         // display result of call to method CountDuplicates
15         Console.WriteLine( "There are {0} words that are duplicates.",
16             CountDuplicates( sentence ) );
17     } // end constructor Duplicates;
18
19     // count number of duplicate words in input sentence
20     public int CountDuplicates( string input )
21     {
22         SortedDictionary< string, int > duplicates =
23             new SortedDictionary< string, int >();
24
25         // break the input up into words and ignore punctuation
26         string[] words = Regex.Split( input, @"[\s\?!.,,]+" );
27
28         // count the number of duplicates of each word with the dictionary
29         foreach ( var word in words )
30         {
31             string token = word.ToLower(); // make the word lower case
32
33             // if key is in dictionary then increment its value by 1
34             // otherwise give it value zero
35             if ( duplicates.ContainsKey( token ) )
36                 ++duplicates[ token ];
37             else
38             {
39                 duplicates.Add( token, 0 );
40             } // end else
41         } // end foreach
42
43         // sum the number of duplicates
44         int sum = 0;
45
46         foreach ( var amount in duplicates.Values )
47             sum += amount;
48     }

```

```

49     return sum;
50 } // end method CountDuplicates
51
52 public static void Main( string[] args )
53 {
54     new Duplicates();
55 } // end Main
56 } // end class Duplicates

```

Please enter a sentence:

How much wood could a woodchuck chuck, if a woodchuck could chuck wood?
 There are 5 words that are duplicates.

23.15 (Using a Generic List) Recall from Fig. 28.2 that class `List` is the generic equivalent of class `ArrayList`. Write an application that inserts 25 random integers from 0 to 100 in order into an object of class `List`. The application should calculate the sum of the elements and the floating-point average of the elements.

ANS:

```

1  // Exercise 23.15 Solution: ListTest.cs
2  // Application inserts and sorts random numbers in a list,
3  // displays the sum, and displays the average.
4  using System;
5  using System.Collections.Generic;
6
7  public class ListTest
8  {
9      public static void Main( string[] args )
10     {
11         List< int > list = new List< int >();
12         int newNumber;
13         Random randomNumber = new Random();
14
15         // Create objects to store in the List
16         for ( int k = 0; k < 25; k++ )
17         {
18             newNumber = randomNumber.Next( 101 );
19             list.Add( newNumber );
20         } // end for
21
22         // sort and output the list
23         list.Sort();
24         Console.WriteLine( "Contents of list:" );
25         foreach ( var number in list )
26             Console.Write( number + " " );
27
28         // calculate total
29         int count = 0;
30         foreach ( var number in list )
31             count += number;
32

```

```

33      Console.WriteLine( "\n\nSum is: {0}\nAverage is: {1:F}", count,
34      ( ( double ) count / list.Count ) );
35  } // end Main
36 } // end class ListTest

```

Contents of list:
3 6 15 16 17 18 32 32 33 40 41 45 45 47 50 61 61 69 78 83 87 90 92 93 94

Sum is: 1248
Average is: 49.92

23.16 (*Reversing a LinkedList*) Write an application that creates a `LinkedList` object of 10 characters, then creates a second list object containing a copy of the first list, but in reverse order.

ANS:

```

1  // Exercise 23.16 Solution: ListReverse.cs
2  // Application creates a linked list, then creates a reverse of the list.
3  using System;
4  using System.Collections.Generic;
5
6  public class ListReverse
7  {
8      // reverses a generic linked list and returns it to the caller
9      public static LinkedList< T > Reverse< T >( LinkedList< T > list )
10     {
11         LinkedList< T > reversed = new LinkedList< T >();
12
13         foreach ( T value in list )
14             reversed.AddFirst( value );
15
16         return reversed;
17     } // end method Reverse
18
19     // output a linked list
20     public static void PrintList< T >( LinkedList< T > list )
21     {
22         foreach ( T value in list )
23             Console.Write( "{0} ", value );
24
25         Console.WriteLine();
26     } // end method PrintList
27
28     public static void Main( string[] args )
29     {
30         // create two linked lists
31         LinkedList< char > list1 = new LinkedList< char >();
32         LinkedList< char > list2 = new LinkedList< char >();
33
34         // use LinkedList insert methods
35         list1.AddFirst( '5' );
36         list1.AddFirst( '@' );
37         list1.AddLast( 'V' );

```

```

38     list1.AddLast( '+' );
39     list1.AddFirst( 'P' );
40     list1.AddFirst( 'c' );
41     list1.AddLast( 'M' );
42     list1.AddLast( '&' );
43     list1.AddFirst( 'y' );
44     list1.AddLast( 'X' );
45
46     Console.WriteLine( "List: " );
47     PrintList( list1 );
48
49     list2 = Reverse( list1 ); // reverse list using method Reverse
50     Console.WriteLine( "Reversed list is: " );
51     PrintList( list2 );
52 } // end Main
53 } // end class ListReverse

```

```

List:
y c P @ 5 V + M & X
Reversed list is:
X & M + V 5 @ P c y

```

23.17 (Prime Numbers and Prime Factorization) Write an application that takes a whole-number input from a user and determines whether it's prime. If the number is not prime, display the unique prime factors of the number. Remember that a prime number's factors are only 1 and the prime number itself. Every number that's not prime has a unique prime factorization. For example, consider the number 54. The prime factors of 54 are 2, 3, 3 and 3. When the values are multiplied together, the result is 54. For the number 54, the prime factors output should be 2 and 3.

ANS:

```

1  // Exercise 23.17 Solution: PrimeFactors.cs
2  // Application finds the prime factors of a number using sets.
3  using System;
4  using System.Collections.Generic;
5  using System.Linq;
6
7  public class PrimeFactors
8  {
9      public PrimeFactors()
10     {
11         // prompt user and get number
12         Console.Write( "Please enter a number, -1 to terminate: " );
13         int inputNumber = Convert.ToInt32( Console.ReadLine() );
14
15         // process user input numbers
16         while ( inputNumber != -1 )
17         {
18             List< int > factorSet = new List< int >();
19             Factorize( inputNumber, factorSet );
20

```

```

21         if ( factorSet.Count == 0 ) // prime number
22             Console.WriteLine(
23                 "    " + inputNumber + " is a prime number." );
24     else
25     {
26         Console.Write( "\n    Prime factors are: " );
27
28         // every key in the sorted dictionary is a factor
29         foreach ( var factor in factorSet.Distinct() )
30             Console.Write( factor + " " );
31     } // end else
32
33     // get next number
34     Console.Write( "\n\nPlease enter a number, -1 to terminate: " );
35     inputNumber = Convert.ToInt32( Console.ReadLine() );
36 } // end while
37 } // end constructor PrimeFactors
38
39 // find prime factors
40 public bool Factorize( int number, List< int > set )
41 {
42     if ( number == 0 || number == 1 )
43         return false;
44
45     // loop through numbers that are less than or equal
46     // to the square root of number
47     for ( int factor = 2; factor <= Math.Sqrt( number ); factor++ )
48     {
49         // has factor
50         if ( number % factor == 0 )
51         {
52             set.Add( factor );
53
54             // just look at what's left over for more factors
55             if ( !Factorize( ( number / factor ), set ) )
56                 set.Add( number / factor );
57
58             return true;
59         } // end if
60     } // end for
61
62     return false;
63 } // end method Factorize
64
65 public static void Main( string[] args )
66 {
67     new PrimeFactors();
68 } // end Main
69 } // end class PrimeFactors

```

Please enter a number, -1 to terminate: 210

Prime factors are: 2 3 5 7

Please enter a number, -1 to terminate: 81

Prime factors are: 3

Please enter a number, -1 to terminate: 1025

Prime factors are: 5 41

Please enter a number, -1 to terminate: -1

23.18 (*Bucket Sort with `LinkedList<int>`*) In Exercise 25.7, you performed a bucket sort of ints by using a two-dimensional array, where each row of the array represented a bucket. If you use a dynamically expanding data structure to represent each bucket, you do not have to write code that keeps track of the number of ints in each bucket. Rewrite your solution to use a one-dimensional array of `LinkedList<int>` buckets.

ANS:

```

1  // Exercise 23.18 Solution: BucketSort.cs
2  // Sort an array's values into ascending order using bucket sort
3  // with a one-dimensional array of LinkedLists.
4  using System;
5  using System.Collections.Generic;
6  using System.Text;
7
8  public class BucketSort
9  {
10     private int[] data; // array of values
11     private static Random generator = new Random();
12
13     // create array of given size and fill with random integers
14     public BucketSort( int size )
15     {
16         data = new int[ size ]; // create space for array
17
18         // fill array with random ints in range 10-99
19         for ( int i = 0; i < size; i++ )
20             data[ i ] = generator.Next( 10, 100 );
21     } // end BucketSort constructor
22
23     // perform bucket sort algorithm on array
24     public void Sort()
25     {
26         // store maximum number of digits in numbers to sort
27         int totalDigits = NumberOfDigits();
28
29         // bucket array where numbers will be placed
30         LinkedList< int >[] pail = new LinkedList< int >[ 10 ];
31

```

```

32     // we must create a LinkedList for each bucket
33     for ( int i = 0; i < 10; i++ )
34         pail[ i ] = new LinkedList< int >();
35
36     // go through all digit places and sort each number
37     // according to digit place value
38     for ( int pass = 1; pass <= totalDigits; pass++ )
39     {
40         DistributeElements( pail, pass ); // distribution pass
41         CollectElements( pail ); // gathering pass
42
43         if ( pass != totalDigits )
44             EmptyBucket( pail ); // set size of buckets to 0
45     } // end for
46 } // end method Sort
47
48 // determine number of digits in the largest number
49 public int NumberOfDigits()
50 {
51     int largest = data[ 0 ]; // set largest to first element
52
53     // loop over elements to find largest
54     foreach ( var element in data )
55         if ( element > largest )
56             largest = element; // set largest to current element
57
58     // calculate number of digits in largest value
59     int digits = ( int ) ( Math.Floor( Math.Log10( largest ) ) + 1 );
60
61     return digits;
62 } // end method NumberOfDigits
63
64 // distribute elements into buckets based on specified digit
65 public void DistributeElements( LinkedList< int >[] pail, int digit )
66 {
67     int bucketNumber; // number of bucket to place element
68
69     // determine the divisor used to get specific digit
70     int divisor = ( int ) ( Math.Pow( 10, digit ) );
71
72     foreach ( var element in data )
73     {
74         // bucketNumber example for hundreds digit:
75         // ( 1234 % 1000 ) / 100 --> 2
76         bucketNumber = ( element % divisor ) / ( divisor / 10 );
77
78         // store element at the end of row bucketNumber
79         pail[ bucketNumber ].AddLast( element );
80     } // end foreach
81 } // end method DistributeElements
82
83 // return elements to original array
84 public void CollectElements( LinkedList< int >[] pails )
85 {

```

```

86     int index = 0; // initialize location in data
87
88     foreach ( var list in pails ) // loop over buckets
89     {
90         // loop over elements in each bucket
91         foreach ( var element in list )
92             data[ index++ ] = element; // add element to array
93     } // end outer foreach
94 } // end method CollectElements
95
96 // set size of all buckets to zero
97 public void EmptyBucket( LinkedList< int >[] pails )
98 {
99     for ( int i = 0; i < 10; i++ )
100         pails[ i ].Clear(); // set size of bucket to 0
101 } // end method EmptyBucket
102
103 // method to output values in array
104 public override string ToString()
105 {
106     StringBuilder temporary = new StringBuilder();
107
108     // iterate through array
109     foreach ( var element in data )
110         temporary.Append( element + " ");
111
112     temporary.AppendLine(); // add newline character
113
114     return temporary.ToString();
115 } // end method ToString
116 } // end class BucketSort

```

```

1 // Exercise 23.18 Solution: BucketSortTest.cs
2 // Test the bucket sort class.
3 using System;
4
5 public class BucketSortTest
6 {
7     public static void Main( string[] args )
8     {
9         // create object to perform bucket sort
10         BucketSort sortArray = new BucketSort( 10 );
11
12         Console.WriteLine( "Before:" );
13         Console.WriteLine( sortArray ); // output unsorted array
14
15         sortArray.Sort(); // sort array
16
17         Console.WriteLine( "After:" );
18         Console.WriteLine( sortArray ); // output sorted array
19     } // end Main
20 } // end class BucketSortTest

```

Before:

57 88 71 98 12 43 90 26 68 41

After:

12 26 41 43 57 68 71 88 90 98



GUI with Windows Presentation Foundation

24

My function is to present old masterpieces in modern frames.

—Rudolf Bing

Instead of being a static one-time event, bonding is a process, a dynamic and continuous one.

—Julius Segal

...they do not declare but only hint.

—Friedrich Nietzsche

Science is the knowledge of consequences, and dependence of one fact upon another.

—Thomas Hobbes

Here form is content, content is form.

—Samuel Beckett

Objectives

In this chapter you'll learn:

- To mark up data using XML.
- To define a WPF GUI with Extensible Application Markup Language (XAML).
- To handle WPF user-interface events.
- To use WPF's commands feature to handle common application tasks such as cut, copy and paste.
- To customize the look-and-feel of WPF GUIs using styles and control templates.
- To use data binding to display data in WPF controls.

Self-Review Exercises

Sections 24.3–24.5

24.1 Which of the following are valid XML element names? (Select all that apply.)

- a) yearBorn
- b) year.Born
- c) year Born
- d) year-Born1
- e) 2_year_born
- f) _year_born_

ANS: a, b, d, f. [Choice c is incorrect because it contains a space. Choice e is incorrect because the first character is a digit.]

24.2 State whether each of the following is *true* or *false*. If *false*, explain why.

- a) XML is a technology for creating markup languages called XML vocabularies.

ANS: True.

- b) XML markup is delimited by forward and backward slashes (/ and \).

ANS: False. In an XML document, markup text is delimited by tags enclosed in angle brackets (< and >) with a forward slash just after the < in the end tag.

- c) All XML start tags must have corresponding end tags.

ANS: True.

- d) Parsers check an XML document's syntax.

ANS: True.

- e) XML does not support namespaces.

ANS: False. XML does support namespaces.

- f) When creating XML elements, document authors must use the set of standard XML tags provided by the W3C.

ANS: False. When creating tags, document authors can use any valid name but should avoid ones that begin with the reserved word `xml` (also `XML`, `Xml`, and so on).

- g) The pound character (#), dollar sign (\$), ampersand (&) and angle brackets (< and >) are examples of XML reserved characters.

ANS: False. XML reserved characters include the ampersand (&), the left-angle bracket (<) and the right-angle bracket (>), but not # and \$.

24.3 In Fig. 19.2, we subdivided the `author` element into more detailed pieces. How might you subdivide the `date` element? Use the date May 5, 2005, as an example.

ANS: `<date>`

`<month>May</month>`

`<day>5</day>`

`<year>2005</year>`

`</date>`.

Sections 24.2 and Sections 24.6–24.15

24.4 State whether each of the following is *true* or *false*. If *false*, explain why.

- a) WPF has GUI, animation, 2D graphics, 3D graphics and multimedia capabilities.

ANS: True.

- b) All of a WPF application's C# code can be replaced by XAML markup.

ANS: False. XAML is designed to be used alongside C#, not to replace it.

- c) You lay out a WPF GUI in the same way you lay out a Windows Forms GUI.

ANS: False. The layout in WPF is primarily flow based, whereas layout in Windows Forms is primarily coordinate based.

d) Events in WPF are the same as they are in Windows Forms.

ANS: False. Events in WPF are routed events.

e) A WPF command can be executed through many user interactions.

ANS: True.

f) If a WPF resource is defined for a Window, then it can be used by any of the Window's child elements.

ANS: True.

g) A WPF style must be defined for a specific control type.

ANS: False. A style may be applied to any type of control.

h) A WPF control template must be defined for a specific control type.

ANS: True.

i) A WPF control's logical and visual trees are the same.

ANS: False. A control's visual tree represents how a control is rendered, whereas a logical tree represents how it's defined.

j) Data bindings in WPF can be specified in XAML by markup extensions.

ANS: True.

24.5 Fill in the blanks in each of the following statements:

a) XAML documents consists of a hierarchy of _____.

ANS: elements.

b) A Window is a(n) _____, meaning it can have exactly one child element or text.

ANS: content control.

c) Properties such as Grid.Row or Canvas.Left are examples of _____ properties.

ANS: attached.

d) MouseLeftButtonDown is a bubbling event. The name of its corresponding tunneling event is _____.

ANS: PreviewMouseLeftButtonDown.

e) When a WPF command is executed, it raises the _____ and _____ events.

ANS: Executed, PreviewExecuted.

f) A menu usually has only _____ and _____ as children.

ANS: MenuItem, Separator.

g) _____ resources are applied at initialization only, whereas _____ resources are reapplied every time the resource is modified.

ANS: Static, dynamic.

h) A WPF control template defines a control's _____ tree.

ANS: visual.

i) To declare a variable that may raise an event, you must add the _____ keyword to its declaration.

ANS: WithEvents.

j) To format the display of items in a ListView, you would use a(n) _____.

ANS: data template.

Exercises

NOTE: Solutions to the programming exercises are located in the So1_ch24 folder. Each exercise has its own folder named ex24_## where ## is a two-digit number representing the exercise number. For example, Exercise 24.6's solution is located in the folder ex24_06.

WPF Graphics and Multimedia

25

Nowadays people's visual imagination is so much more sophisticated, so much more developed, particularly in young people, that now you can make an image which just slightly suggests something, they can make of it what they will.

—Robert Doisneau

In shape, it is perfectly elliptical. In texture, it is smooth and lustrous. In color, it ranges from pale alabaster to warm terra cotta.

—Sydney J Harris, "Tribute to an Egg"

Objectives

In this chapter you'll learn:

- To manipulate fonts.
- To draw basic WPF shapes like Lines, Rectangles, Ellipses and Polygons.
- To use WPF brushes to customize the Fill or Background of an object.
- To use WPF transforms to reposition or reorient GUI elements.
- To completely customize the look of a control while maintaining its functionality.
- To animate the properties of a GUI element.
- To transform and animate 3-D objects.

Self-Review Exercises

25.1 State whether each of the following is *true* or *false*. If *false*, explain why.

a) The unit of measurement for the `FontSize` property is machine independent.

ANS: True.

b) A `Line` is defined by its length and its direction.

ANS: False. A `Line` is defined by a start point and an end point.

c) If an object's `Fill` is not defined, it uses the default `White` color.

ANS: False—When no `Fill` is defined, the object is transparent.

d) A `Polyline` and `Polygon` are the same, except that the `Polygon` connects the first point in the `PointCollection` with the last point.

ANS: True. e

e) A `Collapsed` element occupies space in the window, but it is transparent.

ANS: False. A `Collapsed` object has a `Width` and `Height` of 0.

f) A `MediaElement` is used for audio or video playback.

ANS: True.

g) A `LinearGradientBrush` always defines a gradient that transitions through colors from left to right.

ANS: False. You can define start and end points for the gradient to change the direction of the transitions.

h) A transform can be applied to a WPF UI element to reposition or reorient the graphic.

ANS: True.

i) A `Storyboard` is the main control for implementing animations into the application.

ANS: True.

j) A 3-D object can be manipulated much as a 2-D object would be manipulated.

ANS: True.

25.2 Fill in the blanks in each of the following statements:

a) A(n) _____ control can be used to display text in the window.

ANS: `TextBlock`.

b) A(n) _____ control can apply `Underlines`, `Overlines`, `Baselines` or `Strikethroughs` to a piece of text.

ANS: `TextDecoration`.

c) The _____ property of the `DoubleAnimation` defines the final value taken by the animated property.

ANS: `To`.

d) The four types of transforms are _____, _____, `TranslateTransform` and `RotateTransform`.

ANS: `SkewTransform`, `ScaleTransform`.

e) The _____ property of a `GradientStop` defines where along the transition the corresponding color appears.

ANS: `Offset`.

f) The _____ property of a `Storyboard` defines what property you want to animate.

ANS: `TargetProperty`.

g) A `Polygon` connects the set of points defined in a(n) _____ object.

ANS: `PointCollection`.

h) The `Position` of a `PerspectiveCamera` is defined with a(n) _____ object while the `Direction` is defined with a(n) _____ object.

ANS: `Point3D`, `Vector3D`.

i) The three basic available shape controls are `Line`, _____ and _____.

ANS: `Rectangle`, `Ellipse`.

j) A(n) _____ creates an opacity mapping from a brush and applies it to an element.

ANS: opacity mask.

k) _____ points are used to define the StartPoint and EndPoint of a gradient to reference locations independently of the control's size.


ANS: Logical.

Exercises

NOTE: Solutions to the programming exercises are located in the sol_ch25 folder. Each exercise has its own folder named ex25_## where ## is a two-digit number representing the exercise number. For example, Exercise 25.3's solution is located in the folder ex25_03.

XML and LINQ to XML: Solutions

26



*Like everything metaphysical,
the harmony between thought
and reality is to be found in the
grammar of the language.*

—Ludwig Wittgenstein

*I played with an idea, and grew
willful; tossed it into the air;
transformed it; let it escape and
recaptured it; made it iridescent
with fancy, and winged it with
paradox.*

—Oscar Wilde

Objectives

In this chapter you'll learn:

- To specify and validate an XML document's structure.
- To create and use simple XSL style sheets to render XML document data.
- To use the Document Object Model (DOM) to manipulate XML in C# programs.
- To use LINQ to XML to extract and manipulate data from XML documents.
- To create new XML documents using the classes provided by the .NET Framework.
- To work with XML namespaces in your C# code.
- To transform XML documents into XHTML using class `XslCompiledTransform`.

Self-Review Exercises

26.1 Fill in the blanks for each of the following:

a) _____ embed application-specific information into an XML document.

ANS: Processing instructions.

b) _____ is Microsoft's XML parser used in Internet Explorer.

ANS: MSXML.

c) XSL element _____ writes a DOCTYPE to the result tree.

ANS: xsl:output.

d) XML Schema documents have root element _____.

ANS: schema.

e) XSL element _____ is the root element in an XSL document.

ANS: xsl:stylesheet.

f) XSL element _____ selects specific XML elements using repetition.

ANS: xsl:for-each.

26.2 State whether each of the following is *true* or *false*. If *false*, explain why.

a) XML Schemas are better than DTDs, because DTDs lack a way of indicating what specific type of data (e.g., numeric, text) an element can contain and DTDs are not themselves XML documents.

ANS: True.

b) A DTD cannot indicate that an element is optional.

ANS: False. DTDs specify optional elements using the question mark (?) occurrence indicator, which indicates an element may appear at most once, or the asterisk (*) occurrence indicator, which indicates the element may appear any number of times.

c) Schema is a technology for locating information in an XML document.

ANS: False. XPath is a technology for locating information in an XML document. XML Schema provides a means for type checking XML documents and verifying their validity.

26.3 Write a processing instruction that includes style sheet wap.xsl for use in Internet Explorer.

ANS: `<?xml-stylesheet type = "text/xsl" href = "wap.xsl"?>`

26.4 Write an XPath expression that locates contact nodes in letter.xml (Fig. 16.4).

ANS: `/letter/contact`

26.5 Describe the Elements and Descendants methods used in this chapter.

ANS: The Elements and Descendants methods of XElement are both overloaded—the version that takes no arguments returns all applicable elements, and the version that takes an XName returns only those elements with the given name. The Elements method returns only direct children, while the Descendants method also returns grandchildren, great-grandchildren, and so on. There are also extension methods for IEnumerable<XElement> that return the children or descendants of all elements in the collection.

26.6 Write the C# code necessary to create an XElement with a local name of "name" and a namespace of "http://www.example.com".

ANS: `XNamespace example = "http://www.example.com";
XElement element = new XElement(example + "name");`

Exercises

NOTE: Solutions to the programming exercises are located in the `sol_ch26` folder. Each exercise has its own folder named `ex26_##` where `##` is a two-digit number representing the exercise number. For example, Exercise 26.3's solution is located in the folder `ex26_03`.

Web App Development with ASP.NET: A Deeper Look: Solutions

27

... the challenges are for the designers of these applications: to forget what we think we know about the limitations of the Web, and begin to imagine a wider, richer range of possibilities. It's going to be fun.

—Jesse James Garrett

If any man will draw up his case, and put his name at the foot of the first page, I will give him an immediate reply. Where he compels me to turn over the sheet, he must wait my leisure.

—Lord Sandwich

Objectives

In this chapter you'll learn:

- Web application development using ASP.NET.
- To handle the events from a Web Form's controls.
- To use validation controls to ensure that data is in the correct format before it's sent from a client to the server.
- To maintain user-specific information.
- To create a data-driven web application using ASP.NET and LINQ to SQL.

Self-Review Exercises

27.1 State whether each of the following is *true* or *false*. If *false*, explain why.

- a) An access rule grants or denies access to a particular directory for a specific user or group of users.

ANS: True.

- b) When using controls from the Ajax Control Toolkit, you must include the `ScriptManager` control at the top of the ASPX page.

ANS: False. The `ToolkitScriptManager` control must be used for controls from the Ajax Control Toolkit. The `ScriptManager` control can be used only for the controls in the **Toolbox's AJAX Extensions** tab.

- c) A master page is like a base class in a visual inheritance hierarchy, and content pages are like derived classes.

ANS: True.

- d) A `GridView` automatically enables sorting and paging of its contents.

ANS: False. Checking **Enable Sorting** in the **GridView Tasks** smart-tag menu changes the column headings in the `GridView` to hyperlinks that allow users to sort the data in the `GridView`. Checking **Enable Paging** in the **GridView Tasks** smart-tag menu causes the `GridView` to split across multiple pages.

- e) AJAX web applications make synchronous requests and wait for responses.

ANS: False. That is what traditional web applications do. AJAX web applications can make asynchronous requests and do not need to wait for responses.

27.2 Fill in the blanks in each of the following statements:

- a) A(n) _____ defines common GUI elements that are inherited by each page in a set of _____.

ANS: master page, content pages.

- b) The main difference between a traditional web application and an Ajax web application is that the latter supports _____ requests.

ANS: asynchronous.

- c) The _____ template is a starter kit for a small multi-page website that uses Microsoft's recommended practices for organizing a website and separating the website's style (look-and-feel) from its content.

ANS: **ASP.NET Web Site**.

- d) The _____ allows you to configure various options that determine how your application behaves.

ANS: **Web Site Administration Tool**.

- e) A `LinqDataSource`'s _____ event occurs every time the `LinqDataSource` selects data from its data context, and can be used to implement custom `Select` queries against the data context.

ANS: `Selecting`.

- f) Setting a `DropDownList`'s _____ property to `True` indicates that a postback occurs each time the user selects an item in the `DropDownList`.


ANS: `AutoPostBack`.

- g) Several controls in the Ajax Control Toolkit are _____—components that enhance the functionality of regular ASP.NET controls.

ANS: extenders.

Exercises

NOTE: Solutions to the programming exercises are located in the `sol_ch27` folder. Each exercise has its own folder named `ex27_##` where `##` is a two-digit number representing the exercise number. For example, Exercise 27.3's solution is located in the folder `ex27_03`.



A client is to me a mere unit, a factor in a problem.

—Sir Arthur Conan Doyle

...if the simplest things of nature have a message that you understand, rejoice, for your soul is alive.

—Eleonora Duse

Objectives

In this chapter you'll learn:

- How to create WCF web services.
- How XML, JSON, XML-Based Simple Object Access Protocol (SOAP) and Representational State Transfer Architecture (REST) enable WCF web services.
- The elements that comprise WCF web services, such as service references, service endpoints, service contracts and service bindings.
- How to create a client that consumes a WCF web service.
- How to use WCF web services with Windows and web applications.
- How to use session tracking in WCF web services to maintain state information for the client.
- How to pass user-defined types to a WCF web service.

Self-Review Exercises

28.1 State whether each of the following is *true* or *false*. If *false*, explain why.

- a) The purpose of a web service is to create objects of a class located on a web service host. This class then can be instantiated and used on the local machine.

ANS: False. Web services are used to execute methods on web service hosts. The web service receives the arguments it needs to execute a particular method, executes the method and returns the result to the caller.

- b) You must explicitly create the proxy class after you add a service reference for a SOAP-based service to a client application.

ANS: False. The proxy class is created by Visual C# or Visual Web Developer when you add a Service Reference to your project. The proxy class itself is hidden from you.

- c) A client application can invoke only those methods of a web service that are tagged with the `OperationContract` attribute.

ANS: True.

- d) To enable session tracking in a web-service method, no action is required other than setting the `SessionMode` property to `SessionMode.Required` in the `ServiceContract` attribute.

ANS: True.

- e) Operations in a REST web service are defined by their own unique URLs.

ANS: True.

- f) A SOAP-based web service can return data in JSON format.

ANS: False. A SOAP web service implicitly returns data in XML format.

- g) For a client application to deserialize a JSON object, the client must define a `Serializable` class with public instance variables or properties that match those serialized by the web service.

ANS: True.

28.2 Fill in the blanks for each of the following statements:

- a) A key difference between SOAP and REST is that SOAP messages have data wrapped in a(n) _____.

ANS: envelope.

- b) A WCF web service exposes its methods to clients by adding the _____ and _____ attributes to the service interface.

ANS: `ServiceContract`, `OperationContract`.

- c) Web-service requests are typically transported over the Internet via the _____ protocol.

ANS: HTTP.

- d) To return data in JSON format from a REST-based web service, the _____ property of the `WebGet` attribute is set to _____.

ANS: `ResponseFormat`, `WebMessageFormat.Json`.

- e) _____ transforms an object into a format that can be sent between a web service and a client.

ANS: Serialization.

- f) To parse a HTTP response in XML data format, the client application must import the response's _____.

ANS: namespace.

Exercises

NOTE: Solutions to the programming exercises are located in the `sol_ch28` folder. Each exercise has its own folder named `ex28_##` where `##` is a two-digit number representing the exercise number. For example, Exercise 28.3's solution is located in the folder `ex28_03`.

Silverlight and Rich Internet Applications: Solutions

29

*Had I the heavens' embroidered
cloths, Enwrought with gold
and silver light.*

—William Butler Yeats

*This world is but a canvas to
our imaginations.*

—Henry David Thoreau

*Something deeply hidden had to
be behind things.*

—Albert Einstein

*Individuality of expression is the
beginning and end of all art.*

—Johann Wolfgang von Goethe

Objectives

In this chapter you'll learn:

- How Silverlight relates to WPF.
- To use Silverlight controls to create Rich Internet Applications.
- To create custom Silverlight controls.
- To use animation for enhanced GUIs.
- To display and manipulate images.
- To use Silverlight with Flickr's web services to build an online photo-searching application.
- To create Silverlight deep zoom applications.
- To include audio and video in Silverlight applications.

Self-Review Exercises

- 29.1** Say whether the statement is *true* or *false*. If it is *false*, explain why.
- a) Silverlight employs all of the same functionality as WPF but in the form of an Internet application.
ANS: False—Silverlight is a subset of WPF, therefore it does not contain all of the same functionality as a WPF application.
 - b) Silverlight competes with RIA technologies such as Adobe Flash and Flex and Sun's JavaFX, and complements Microsoft's ASP.NET and ASP.NET AJAX.
ANS: True
 - c) The .xap file contains the application and its supporting resources and is packaged by the IDE.
ANS: True
 - d) Silverlight's template control is Window.
ANS: False—Unlike WPF applications, the template control for Silverlight applications is the UserControl.
 - e) Users can create custom controls by using the Silverlight Style and ControlTemplate controls.
ANS: False—While Styles and ControlTemplates can be used to customize existing controls, UserControl is the template used to create custom controls.
 - f) When you call WebClient's DownloadStringAsync method, the user can still interact with the application while the string is downloading.
ANS: True
 - g) A deep zoom image is just a high-resolution image.
ANS: False—A deep zoom image is really a collection of images. Deep Zoom Composer separates your original collage into these images, which are sent over the Internet to the client machine.
- 29.2** Fill in the blanks with the appropriate answer.
- a) The three basic animation controls are _____, _____, and _____.
ANS: DoubleAnimation, PointAnimation, ColorAnimation.
 - b) An object of class _____ can be used to invoke a web service.
ANS: WebClient.
 - c) The XDocument method _____ converts a String containing XML into an object that can be used with LINQ to XML.
ANS: Parse.
 - d) Namespace _____ is required to use LINQ to XML in your application.
ANS: System.Xml.Linq.
 - e) When a MediaElement has finished playing, it is in the _____ state.
ANS: Paused.
 - f) The three layout controls for Silverlight are _____, _____ and _____.
ANS: Grid, StackPanel, Canvas.
 - g) The _____ of a MultiScaleImage represents the area of the deep zoom image that the user is currently viewing.
ANS: viewport.

Exercises

NOTE: Solutions to the programming exercises are located in the sol_ch29 folder. Each exercise has its own folder named ex29_## where ## is a two-digit number representing the exercise number. For example, Exercise 29.3's solution is located in the folder ex29_03.