

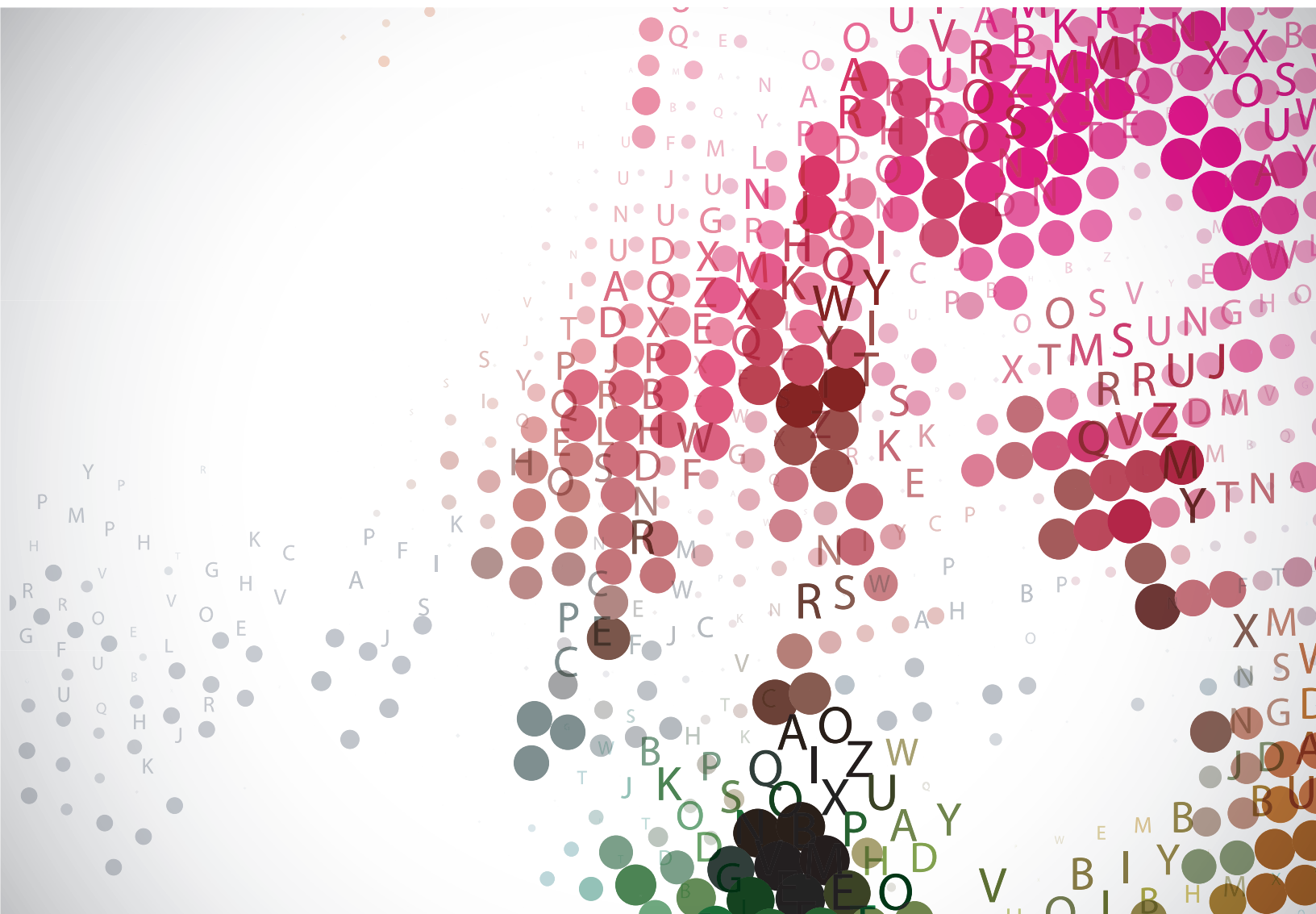
INTERNATIONAL
EDITION



An Introduction To Programming **Using Visual Basic® 2012**

NINTH EDITION

David I. Schneider



ALWAYS LEARNING

PEARSON



get with the programming

Through the power of practice and immediate personalized feedback, MyProgrammingLab improves your performance.

MyProgrammingLabTM

Learn more at www.myprogramminglab.com

**AN INTRODUCTION
TO PROGRAMMING USING**
VISUAL BASIC® 2012
NINTH EDITION

**AN INTRODUCTION
TO PROGRAMMING USING**
VISUAL BASIC[®] 2012

NINTH EDITION

David I. Schneider

University of Maryland

International Edition contributions by

B.R. Chandavarkar

NITK Surathkal

Prentice Hall

Boston Columbus Indianapolis New York San Francisco Upper Saddle River
Amsterdam Cape Town Dubai London Madrid Milan Munich Paris Montreal Toronto
Delhi Mexico City Sao Paulo Sydney Hong Kong Seoul Singapore Taipei Tokyo

Editorial Director, ECS: Marcia Horton
Executive Editor: Tracy Johnson
Editorial Assistant: Jenah Blitz-Stoehr
Director of Marketing: Christy Lesko
Marketing Manager: Yezan Alayan
Senior Marketing Coordinator: Kathryn Ferranti
Director of Production: Erin Gregg
Senior Managing Editor: Scott Disanno
Production Project Manager: Kayla Smith-Tarbox
Publisher, International Edition: Angshuman Chakraborty
Publishing Administrator and Business Analyst,
International Edition: Shokhi Shah Khandelwal
Associate Print and Media Editor, International Edition:
Anuprova Dey Chowdhuri
Acquisitions Editor, International Edition: Shivangi Ramachandran

Publishing Administrator, International Edition: Hema Mehta
Project Editor, International Edition: Karthik Subramanian
Senior Manufacturing Controller, Production,
International Edition: Trudy Kimber
Manufacturing Buyer: Lisa McDowell
Art Director: Anthony Gemmellaro
Cover Designer: Anthony Gemmellaro
Manager, Rights and Permissions: Michael Joyce
Text Permission Coordinator: Anna Waluk /
Electronic Publishing Services
Cover Image: © Redshinestudio / Shutterstock
Media Project Manager: Renata Butera
Full-Service Project Management: Laserwords
Cover Printer: © Redshinestudio / Shutterstock

Pearson Education Limited
Edinburgh Gate
Harlow
Essex CM20 2JE
England

and Associated Companies throughout the world

Visit us on the World Wide Web at:
www.pearsoninternationaleditions.com

© Pearson Education Limited 2014

The rights of David I. Schneider to be identified as author of this work have been asserted by him in accordance with the Copyright, Designs and Patents Act 1988.

Authorized adaptation from the United States edition, entitled An Introduction to Programming Using Visual Basic 2012, Ninth Edition, ISBN 978-0-13-337850-4, by David I. Schneider, published by Pearson Education © 2014.

All rights reserved. No part of this publication may be reproduced, stored in a retrieval system, or transmitted in any form or by any means, electronic, mechanical, photocopying, recording or otherwise, without either the prior written permission of the publisher or a license permitting restricted copying in the United Kingdom issued by the Copyright Licensing Agency Ltd, Saffron House, 6–10 Kirby Street, London EC1N 8TS.

All trademarks used herein are the property of their respective owners. The use of any trademark in this text does not vest in the author or publisher any trademark ownership rights in such trademarks, nor does the use of such trademarks imply any affiliation with or endorsement of this book by such owners.

Microsoft and/or its respective suppliers make no representations about the suitability of the information contained in the documents and related graphics published as part of the services for any purpose. All such documents and related graphics are provided “as is” without warranty of any kind. Microsoft and/or its respective suppliers hereby disclaim all warranties and conditions with regard to this information, including all warranties and conditions of merchantability, whether express, implied or statutory, fitness for a particular purpose, title and non-infringement. In no event shall Microsoft and/or its respective suppliers be liable for any special, indirect or consequential damages or any damages whatsoever resulting from loss of use, data or profits, whether in an action of contract, negligence or other tortious action, arising out of or in connection with the use or performance of information available from the services.

The documents and related graphics contained herein could include technical inaccuracies or typographical errors. Changes are periodically added to the information herein. Microsoft and/or its respective suppliers may make improvements and/or changes in the product(s) and/or the program(s) described herein at any time. Partial screen shots may be viewed in full within the software version specified.

Microsoft® and Windows® are registered trademarks of the Microsoft Corporation in the U.S.A. and other countries. This book is not sponsored or endorsed by or affiliated with the Microsoft Corporation.

ISBN 10: 0-273-79334-9
ISBN 13: 978-0-273-79334-2

British Library Cataloguing-in-Publication Data

A catalogue record for this book is available from the British Library

10 9 8 7 6 5 4 3 2 1
14 13 12 11 10

Typeset in Goudy by Laserwords

Printed and bound by Courier Kendallville in The United States of America
The publisher’s policy is to use paper manufactured from sustainable forests.

PEARSON

Guide to VideoNotes

www.pearsoninternationaleditions.com/schneider



VideoNote

Chapter 2	Visual Basic, Controls, and Events	
	Visual Basic Controls	43
	Basic Controls, Sizing and Aligning	52
	Using Multiple Controls	53
	Moving a Textbox (Homework)	57
	Event Procedures	58
Chapter 3	Variables, Input, and Output	
	Numbers	78
	Option Explicit and Option Strict	93
	Widening/Narrowing, Comments and Scope	98
	Variable Scope	100
	Formatting Output	109
	Input Boxes and Message Boxes	114
	Loan Calculator (Home work)	124
Chapter 4	Decisions	
	Relational and Logical Operators	127
	If Blocks	134
	Select Case Blocks	156
	Grading System Problem (Homework)	167
	Listboxes, Radio Buttons, and Checkboxes for Input	171
Chapter 5	General Procedures	
	Function Procedures	188
	Sub Procedures	204
	Scope and Lifetime of Variables	222
	Debugging Functions and Sub Procedures	223
	Hardware Store Application (Homework)	241
Chapter 6	Repetition	
	Do Loops	246
	For . . . Next Loops	260
	Nested For . . . Next Loops	264
	List Boxes and Loops	274
	Sieve of Eratosthenes (Homework)	289
Chapter 7	Arrays	
	Declaring and Using Arrays	293
	Array Methods	297
	For Each Loops	300
	Copying an Array, Split Method, and Join Function	303
	LINQ	318
	Arrays of Structures	332
	Two-Dimensional Arrays	357

Chapter 8	Text Files	
	Managing Text Files	392
	StreamReaders and StreamWriters	406
	Exception Handling	413
	XML	424
	DNA sequence data (Homework)	442
Chapter 9	Additional Controls and Objects	
	List Boxes and Combo Boxes	448
	Timer, PictureBox, Menustrip, and Scrollbar Controls; Random Class	455
	Multiple-Form Programs	470
	Graphics	482
Chapter 10	Databases	
	Introduction to Databases	504
	Querying Tables	511
	Editing Databases	530
	Richard's Catering Case Study (Homework)	539
Chapter 11	Object-Oriented Programming	
	Classes and Objects	542
	Arrays of Objects	559
	Inheritance	570
	Student Registration Application (Homework)	586

Guide to Application Topics

Business and Economics

Admission fee, 174
Airline reservations, 385, 500
Analyze a Loan case study, 371
Analyze fuel economy, 388
Analyze growth of chains, 366
Annuity, 89, 203, 258, 259, 272
APY, 153
Automated directory assistance, 387
Automobile depreciation, 271
Bank account, 588
Break-even analysis, 88, 167
Business travel expenses, 502
Calculate a profit, 88, 138, 201
Calculate a tip, 149, 217
Calculate weekly pay, 149, 193, 228, 476, 557
Car loan, 184, 258, 272
Cash register, 557, 568, 586
Cash reward, 168
Change from a sale, 150
Checking account, 588
 transactions, 480
Compare interest rates, 153
Compare two salary options, 272
Compound interest, 89, 153, 177, 193, 203, 250, 257, 258, 259, 271, 480
Computers in the workplace, 365
Consumer options, 169
Consumer price index, 258
Cost of a computer system, 178
Cost of a tour, 167
Cost of benefits, 175, 177
Cost of electricity, 105
Cost of flash drives, 181
Create sales receipt, 422
Credit card account, 227, 481
Crop production, 89, 274
Currency exchange rates, 524
Depreciation, 271, 287
Discounted price, 88, 104
Display economic data in a bar chart, 273, 487, 494
Display economic data in a pie chart, 485, 494, 496
Doubling time of an investment, 119, 257
Employee paycheck receipt, 569
Estate tax, 119
FICA tax, 140, 232, 558
Fixed cost, 167
Future value, 108, 193
Gather billing information, 481
Generate an order form, 241
Growth of an investment, 203
Income tax, 152, 180
ISBN code, 381
Itemized bill, 123, 240
Lifetime earnings, 271
Loan analysis, 124, 480
Loan calculator, 242
Mail-order company, 539
Maintain a membership list, 498
Manage telephone directories, 441
Marginal revenue and cost, 167
Membership fee, 181
Minimum wage, 494
Monetary units of countries, 513, 518
Mortgage, 227, 258
Number of restaurants in U.S., 90
Pay raise, 227
Payroll, 232, 476, 587
Percentage profit, 89
Postage costs, 202
Present value, 108
Price-to-earnings ratio, 106
Recording Checks and Deposits case study, 431
Rental costs, 183
Restaurant order, 184, 568
Retirement plan, 180
Revenue, 167
Rule of '72', 285
Sales commission, 108
Sales tax, 119, 457
Savings plan, 258
Simple interest, 271
Simulate a lottery, 456
Stock purchase, 88
Subscriber data, 404
Supply and demand, 274
Tax return, 180
Total cost, 149
Total income, 473
Total salaries paid, 369
Track inventory, 365, 499, 586
Universal Product Code, 442
Weekly Payroll case study, 232
Withdrawal from a savings account, 150
Withholding tax, 233, 569

General Interest

Age of a tire, 168
 American Heart Assn recommendation, 183
 Anagram, 330
 Analyze Shakespeare sonnet, 314
 Animation, 488, 497
 Bachelor degrees conferred, 382
 Birthdays, 152
 Body Mass Index, 201
 Caffeine absorption, 285
 Calculate age, 112, 115, 121, 153
 Chain-link sentence, 318
 Cloudiness descriptors, 166
 College admissions, 185
 College majors, 495
 Colleges, 337, 340, 429
 Computer pioneers, 353
 Convert temperatures, 189, 469
 Crayola crayons, 316, 405
 Declaration of Independence, 120
 Determine day of week, 121
 Digital clock, 467
 Distance between cities, 358
 Distance from a storm, 105
 Flags, 470, 493, 497
 Freshman life goals, 495
 Friday the 13th, 273
 Game of Life, 386
 Grade book, 540
 Ideal weight, 271
 Language translation, 383
 Leap years, 150, 203
 Military time, 151
 Monthly precipitation, 367
 Movies, 151, 173, 247, 528, 529, 537
 Nutritional content of foods, 359
 Old McDonald had a farm, 216
 Palindrome, 288, 317
 Physician's abbreviations, 168
 Pig Latin, 150
 Pizza consumption, 90
 Population growth, 90, 257
 Population of cities, 512–117, 521, 522, 534, 537
 Presidential eligibility, 179
 Principal languages, 496
 Proverbs, 241
 Qwerty words, 272
 Radioactive decay, 258, 270
 Rating of hurricanes, 201
 Spread of an epidemic, 501
 Stopwatch, 455
 Supreme Court justices, 353, 354, 393, 396, 405, 428
 The Twelve Days of Christmas, 355
 Times Square ball, 469
 Training heart rate, 105, 201
 U.S. cities, 349
 U.S. presidents, 169, 297, 321, 331, 393, 404, 454
 U.S. Senate, 430, 440
 U.S. states, 275, 282–84, 302, 323, 351, 398, 428
 United Nations, 332, 335, 336, 523, 524
 University rankings, 366
 Voting machine, 499
 Weather beacon, 137

Mathematics

Areas of geometric shapes, 167, 579
 Calculate a median, 331, 367
 Calculate a range, 256, 284, 315
 Calculate a spread, 586
 Calculate a sum, 270, 313, 314, 315, 423
 Calculate an average, 90, 107, 150, 217, 248, 273, 277, 301, 314, 315, 317, 330, 331, 341, 367
 Calculate population densities, 351
 Calculator, 123, 572, 586
 Coefficient of restitution, 255
 Convert temperatures, 255
 Convert units of length, 124, 380, 453
 Calculate with fractions, 557, 568
 Curve grades, 381
 Factorial, 273
 Factorization, 258
 Greatest common divisor, 257
 ISBN codes, 381
 Magic squares, 369
 Measurements on a square, 556, 568
 Projectile motion, 90, 286
 Quadratic equation, 184
 Standard deviation, 284, 315, 381
 Surface area, 200

Sports and Games

Baseball, 106, 352, 430, 440, 443, 501, 525–6
 Blackjack, 589
 Carnival game, 468
 Dice, 469, 470, 556, 567, 568
 Football, 280, 281, 294, 299, 330
 Golf, 329, 367
 Pick-up-sticks, 242
 Poker, 384, 467, 563
 Soccer league, 384
 Triathlon, 105

CONTENTS

Guide to VideoNotes 7

Guide to Application Topics 9

Preface 15

Acknowledgments 19

Using this Book for a Short or Condensed Course 21

Chapter 1 An Introduction to Computers and Problem Solving 23

1.1 An Introduction to Computing and Visual Basic 24

1.2 Program Development Cycle 27

1.3 Programming Tools 29

Chapter 2 Visual Basic, Controls, and Events 37

2.1 An Introduction to Visual Basic 2012 38

2.2 Visual Basic Controls 40

2.3 Visual Basic Events 58

Summary 74

Chapter 3 Variables, Input, and Output 75

3.1 Numbers 76

3.2 Strings 91

3.3 Input and Output 109

Summary 122

Programming Projects 123

Chapter 4 Decisions 125

4.1 Relational and Logical Operators 126

4.2 If Blocks 134

4.3 Select Case Blocks 155

4.4 Input via User Selection 170

Summary 182

Programming Projects 183

Chapter 5 General Procedures 187

5.1 Function Procedures 188

5.2 Sub Procedures, Part I 203

5.3 Sub Procedures, Part II 218

5.4 Modular Design 228

5.5 A Case Study: Weekly Payroll 232

Summary 240

Programming Projects 240

Chapter 6 Repetition 245

6.1 Do Loops 246

6.2 For . . . Next Loops 259

6.3 List Boxes and Loops 274

Summary 285

Programming Projects 285

Chapter 7 Arrays 291

7.1 Creating and Using Arrays 292

7.2 Using LINQ with Arrays 318

- 7.3 Arrays of Structures 332
- 7.4 Two-Dimensional Arrays 357
- 7.5 A Case Study: Analyze a Loan 371

Summary 379

Programming Projects 380

Chapter 8 Text Files 389

- 8.1 Managing Text Files 390
- 8.2 StreamReaders, StreamWriters, and Structured Exception Handling 406
- 8.3 XML 424
- 8.4 A Case Study: Recording Checks and Deposits 431

Summary 439

Programming Projects 440

Chapter 9 Additional Controls and Objects 445

- 9.1 List Boxes and Combo Boxes 446
- 9.2 Eight Additional Controls and Objects 455
- 9.3 Multiple-Form Programs 470
- 9.4 Graphics 482

Summary 497

Programming Projects 498

Chapter 10 Databases 503

- 10.1 An Introduction to Databases 504
- 10.2 Editing and Designing Databases 530

Summary 538

Programming Projects 539

Chapter 11 Object-Oriented Programming **541**

11.1 Classes and Objects 542

11.2 Working with Objects 559

11.3 Inheritance 570

Summary 587

Programming Projects 588

Appendices **591**

Appendix A ANSI Values 591

Appendix B How To 593

Appendix C Files and Folders 605

Appendix D Visual Basic Debugging Tools 607

Answers **617**

Index **673**

PREFACE

Visual Basic has been a widely used programming language since its introduction in 1991. Its latest incarnation, Visual Basic 2012, brings continued refinement of the language. Visual Basic programmers are enthusiastically embracing the powerful capabilities of the language. Likewise, students learning their first programming language will find VB 2012 the ideal tool to understand the development of computer programs.

My objectives when writing this text were as follows:

1. *To develop focused chapters.* Rather than covering many topics superficially, I concentrate on important subjects and cover them thoroughly.
2. *To use examples and exercises with which students can relate, appreciate, and feel comfortable.* I frequently use real data. Examples do not have so many embellishments that students are distracted from the programming techniques illustrated.
3. *To produce compactly written text that students will find both readable and informative.* The main points of each topic are discussed first and then the peripheral details are presented as comments.
4. *To teach good programming practices that are in step with modern programming methodology.* Problem solving techniques and structured programming are discussed early and used throughout the book. The style follows object-oriented programming principles.
5. *To provide insights into the major applications of computers.*

What's New in the Ninth Edition

Among the changes in this edition, the following are the most significant.

1. **Visual Basic Upgraded** The version of Visual Basic has been upgraded from Visual Basic 2010 to Visual Basic 2012, and relevant new features of Visual Basic 2012 have been addressed.
2. **Additional Exercises** We have added 50 new exercises.
3. **Updated Data** We have updated the real-world data appearing in exercises, examples, and data files.
4. **Discussion of Printing Moved** The discussion of printing has been moved from Chapter 3 to Chapter 9.
5. **Formatting Statements Changed** The functions used to format strings, numbers, and dates have been replaced with the ToString method.
6. **Captions** Every example and applied exercise has been labeled with a caption identifying its type of application.
7. **Screen Captures** Output for most applied exercises and programming projects are shown in screen captures. This feature helps clarify the intent of each exercise.
8. **Windows 8** The screen captures have been updated from Windows 7 to Windows 8 captures.

Unique and Distinguishing Features

Exercises for Most Sections. Each section that teaches programming has an exercise set. The exercises both reinforce the understanding of the key ideas of the section and challenge the student to explore applications. Most of the exercise sets require the student to trace programs, find errors, and write programs. The answers to all the odd-numbered exercises in Chapters 2 through 7 and the short-answer odd-numbered exercises from Chapters 8, 9, 10, and 11 are given at the end of the text. A screen capture accompanies most programming answers.

Practice Problems. Practice Problems are carefully selected exercises located at the end of a section, just before the exercise set. Complete solutions are given following the exercise set. The practice problems often focus on points that are potentially confusing or are best appreciated after the student has thought about them. The reader should seriously attempt the practice problems and study their solutions before moving on to the exercises.

Programming Projects. Beginning with Chapter 3, every chapter contains programming projects. The programming projects not only reflect the variety of ways that computers are used in the business community, but also present some games and general-interest topics. The large number and range of difficulty of the programming projects provide the flexibility to adapt the course to the interests and abilities of the students. Some programming projects in later chapters can be assigned as end-of-the-semester projects.

Comments. Extensions and fine points of new topics are deferred to the “Comments” portion at the end of each section so that they will not interfere with the flow of the presentation.

Case Studies. Each of the three case studies focuses on an important programming application. The problems are analyzed and the programs are developed with top-down charts and pseudocode. The programs can be downloaded from the companion website at www.pearsoninternationaleditions.com/schneider.

Chapter Summaries. In Chapters 2 through 11, the key results are stated and the important terms are summarized at the end of the chapter.

“How To” Appendix. Appendix B provides a compact, step-by-step reference on how to carry out standard tasks in the Visual Basic and Windows environments.

Appendix on Debugging. The placing of the discussion of Visual Basic’s sophisticated debugger in Appendix D allows the instructor flexibility in deciding when to cover this topic.

Guide to Application Topics. This section provides an index of programs that deal with various topics including Business, Mathematics, and Sports.

VideoNotes. Nearly 50 VideoNotes are available at www.pearsoninternationaleditions.com/schneider. VideoNotes are Pearson’s visual tool designed for teaching key programming concepts and techniques. VideoNote icons are placed in the margin of the text book to notify the reader when a topic is discussed in a video. Also, a Guide to Video Notes summarizing the different videos throughout the text is included.

Solution Manuals. The Student Solutions Manual contains the answer to every odd-numbered exercise. The Instructor Solutions Manual contains the answer to every

exercise and programming project. Both solution manuals are in pdf format and can be downloaded from the Publisher's Web site.

Source Code. The programs for all examples and case studies can be downloaded from the Publisher's Web site.

How to Access Instructor and Student Resource Materials

Online Practice and Assessment with **MyProgrammingLab™**

MyProgrammingLab helps students fully grasp the logic, semantics, and syntax of programming. Through practice exercises and immediate, personalized feedback, MyProgrammingLab improves the programming competence of beginning students who often struggle with the basic concepts and paradigms of popular high-level programming languages.

A self-study and homework tool, a MyProgrammingLab course consists of hundreds of small practice problems organized around the structure of this textbook. For students, the system automatically detects errors in the logic and syntax of their code submissions and offers targeted hints that enable students to figure out what went wrong—and why. For instructors, a comprehensive gradebook tracks correct and incorrect answers and stores the code inputted by students for review.

For a full demonstration, to see feedback from instructors and students, or to get started using MyProgrammingLab in your course, visit www.myprogramminglab.com.

Instructor Resources

The following protected instructor resource materials are available on the Publisher's Web site at www.pearsoninternationaleditions.com/schneider. For username and password information, please contact your local Pearson representative.

- Test Item File
- PowerPoint Lecture Slides
- Instructor Solutions Manual
- VideoNotes
- Programs for all examples, case studies, and answers to exercises and programming projects (Databases, text files, and picture files needed for the exercises are included in the Programs folder.)

Student Resources

Access to the Premium Website and VideoNotes tutorials is located at www.pearsoninternationaleditions.com/schneider. Students must use the access card located in the front of the book to register and access the online material. Instructors must register on the site to access the material.

The following content is available through the Premium Web site:

- VideoNotes
- Student Solutions Manual
- Programs for examples and case studies (Databases, text files, and picture files needed for the exercises are included in the Programs folder.)

ACKNOWLEDGMENTS

Many talented instructors and programmers provided helpful comments and constructive suggestions during the many editions this text and I am most grateful for their contributions. The current edition benefited greatly from the valuable comments of the following reviewers:

Chris Olson, Dakota State University
Douglas B. Bock, Southern Illinois University
Gary E. Sullivan, Weatherford College
J.T. Shim, Louisiana Tech University
Laurence Boxer, Niagara University and State University of New York at Buffalo

Many people are involved in the successful publication of a book. I wish to thank the dedicated team at Pearson whose support and diligence made this textbook possible, especially Jenah Blitz-Stoehr, Editorial Assistant for Computer Science, and Scott Disanno, Senior Managing Editor.

I would like to thank Sharon O'Donnell and Craig Cornell for their excellent proofreading. Production editor Kayla Smith-Tarbox did a fantastic job producing the book and keeping it on schedule. I am grateful to John Russo of the Wentworth Institute of Technology for producing the VideoNotes that accompany the book. The skill and graciousness of the team at Laserwords made for a pleasant production process.

I extend special thanks to my editor Tracy Johnson. Her ideas and enthusiasm helped immensely with the preparation of the book.

David I. Schneider

The publishers wish to thank Arup Kumar Bhattacharjee and Soumen Mukherjee of RCC Institute of Information Technology, Kolkata for reviewing the content of the International Edition.

USING THIS BOOK FOR A SHORT OR CONDENSED COURSE

This book provides more than enough material for a complete semester course. For a course shorter than a semester in length, it will be necessary to bypass some sections. The following syllabus provides one possible way to present an abbreviated introduction to programming.

Chapter 1 An Introduction to Computers and Problem Solving

- 1.1 An Introduction to Computing and Visual Basic

Chapter 2 Visual Basic, Controls, and Events

- 2.1 An Introduction to Visual Basic 2012
- 2.2 Visual Basic Controls
- 2.3 Visual Basic Events

Chapter 3 Variables, Input, and Output

- 3.1 Numbers
- 3.2 Strings
- 3.3 Input and Output

Chapter 4 Decisions

- 4.1 Relational and Logical Operators
- 4.2 If Blocks
- 4.3 Select Case Blocks
- 4.4 Input via User Selection

Chapter 5 General Procedures¹

- 5.1 Function Procedures
- 5.2 Sub Procedures, Part I

Chapter 6 Repetition

- 6.1 Do Loops
- 6.2 For . . . Next Loops

Chapter 7 Arrays

- 7.1 Creating and Accessing Arrays
- 7.2 Using LINQ with Arrays

Chapter 8 Text Files²

- 8.1 Managing Text Files
- or 8.2 StreamReaders, StreamWriters, Structured Exception Handling

¹Passing by reference can be omitted or just mentioned briefly. In Chapters 6 through 11, ByRef is used only in Example 6 of Section 7.3 (Arrays of Structures) and in the Chapter 7 case study. In both of those programs, it is used to obtain input.

²Sections 8.1 and 8.2 are independent of each other.

1

An Introduction to Computers and Problem Solving



1.1 An Introduction to Computing and Visual Basic 24

1.2 Program Development Cycle 27

- ◆ Performing a Task on the Computer ◆ Program Planning

1.3 Programming Tools 29

- ◆ Flowcharts ◆ Pseudocode ◆ Hierarchy Chart ◆ Decision Structure
- ◆ Direction of Numbered NYC Streets Algorithm ◆ Repetition Structure
- ◆ Class Average Algorithm

1.1 An Introduction to Computing and Visual Basic

An Introduction to Programming Using Visual Basic 2012 is about problem solving using computers. The programming language used is Visual Basic 2012 (hereafter shortened to Visual Basic), but the principles apply to most modern programming languages. Many of the examples and exercises illustrate how computers are used in the real world. Here are some questions that you may have about computers and programming.

Question: *How do we communicate with the computer?*

Answer: Many languages are used to communicate with the computer. At the lowest level, there is *machine language*, which is understood directly by the microprocessor but is difficult for humans to understand. Visual Basic is an example of a *higher-level language*. It consists of instructions to which people can relate, such as Click, If, and Do. Some other well-known higher-level languages are Java, C++, and Python.

Question: What is a GUI?

Answer: GUI (pronounced GOO-ee) stands for “graphical user interface”. Both Windows and Visual Basic use a graphical user interface; that is, they employ objects such as windows, icons, and menus that can be manipulated by a mouse. Non-GUI-based programs use only text and are accessed solely via a keyboard.

Question: *How do we get computers to perform complicated tasks?*

Answer: Tasks are broken down into a sequence of instructions that can be expressed in a computer language (this text uses Visual Basic). A sequence of instructions is called a *program*. Programs can range in size from two or three instructions to millions of instructions. The process of executing the instructions is called *running* the program.

Question: *What are the meanings of the terms “programmer” and “user”?*

Answer: A *programmer* (sometimes also called a *developer*) is a person who solves problems by writing programs on a computer. After analyzing the problem and developing a plan for solving it, the programmer writes and tests the program that instructs the computer how to carry out the plan. The program might be run many times, either by the programmer or by others. A *user* is any person who runs a program. While working through this text, you will function both as a programmer and as a user.

Question: *Are there certain characteristics that all programs have in common?*

Answer: Most programs do three things: take in data, manipulate data, and produce desired results. These operations are referred to as *input*, *processing*, and *output*. The input data might be held in a portion of the program, reside on a disk drive, or be provided by the user in response to requests made by the computer while the program is running. The processing of the input data occurs inside the computer and can take from a fraction of a second to many hours. The output data are either displayed on a monitor, printed on a printer, or recorded on a disk. As a simple example, consider a program that computes sales tax. An item of input data is the cost of the thing purchased. The processing consists of multiplying the cost by a certain percentage. An item of output data is the resulting product, the amount of sales tax to be paid.

Question: *Many programming languages, including Visual Basic, use a zero-based numbering system. What is a zero-based numbering system?*

Answer: In a zero-based numbering system, counting begins with zero instead of one. For example, in the word “code”, “c” would be the zeroth letter, “o” would be the first letter, and so on.

Question: What are the meanings of the terms “hardware” and “software”?

Answer: Hardware refers to the physical components of the computer, including all peripherals, the central processing unit, disk drives, and all mechanical and electrical devices. Programs are referred to as software.

Question: How are problems solved with a program?

Answer: Problems are solved by carefully reading them to determine what data are given and what outputs are requested. Then a step-by-step procedure is devised to process the given data and produce the requested output.

Question: How did Visual Basic 2012 evolve?

Answer: In the early 1960s, two mathematics professors at Dartmouth College developed a programming language called BASIC to provide their students with an easily learned language that could tackle complicated programming projects. As the popularity of BASIC grew, refinements were introduced that permitted structured programming, which increased the reliability of programs. Visual Basic 1.0 is a graphical version of BASIC developed in 1991 by the Microsoft Corporation to allow easy, visual-oriented development of Windows applications. Visual Basic 2012 is a more advanced and powerful version of the original Visual Basic.

Question: Are there any prerequisites to learning Visual Basic 2012?

Answer: Since Visual Basic is used to write Windows applications, you should be familiar with Windows and understand how folders and files are managed with Windows. The key concepts for understanding folders and files are presented in Appendix C.

Question: Will it matter whether Windows Vista, Windows 7, or Windows 8 are used as the underlying operating system?

Answer: Visual Basic runs fine with all three of these versions of Windows. However, the windows will vary slightly in appearance. Figure 1.1 shows the appearance of a typical window produced in Visual Basic with each of the three versions of Windows. The appearance of windows in Windows 7 and 8 depends on the Windows product edition, the hardware on your system, and your own personal preferences. Most likely your windows will look like the ones in Fig. 1.1(b) and (c). In this book, all screen captures have been done with the Windows 8 operating system.



FIGURE 1.1 Visual Basic windows.

Question: What is an example of a program developed in this textbook?

Answer: Figure 1.2 shows a program from Chapter 7 when it is first run. After the user types in a first name and clicks on the button, the names of the presidents who have that first name are displayed. Figure 1.3 shows the output.

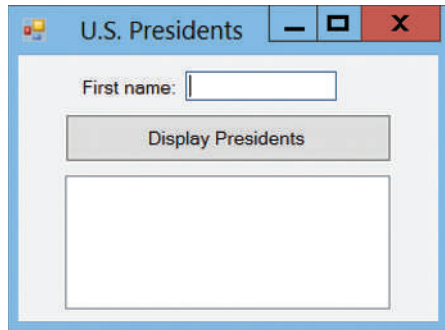


FIGURE 1.2 Window when program is first run.

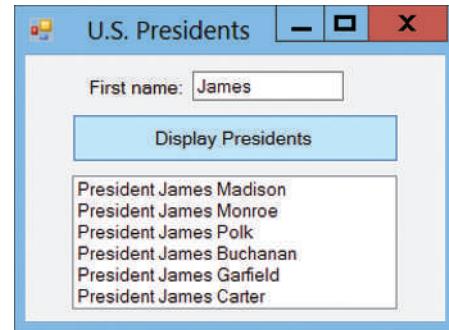


FIGURE 1.3 Window after a name is entered and the button is clicked.

Question: How does the programmer create the aforementioned program?

Answer: The programmer begins with a blank window called a **form**. See Fig. 1.4. The programmer adds objects, called **controls**, to the form and sets properties for the controls. In Fig. 1.5, four controls have been placed on the form. The Text properties of the form, the label, and the button have been set to “U.S. Presidents”, “First name:”, and “Display Presidents”. The Name property of the list box was set to “lstPres”.

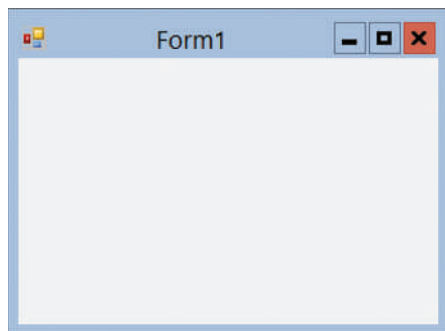


FIGURE 1.4 A blank Visual Basic form.

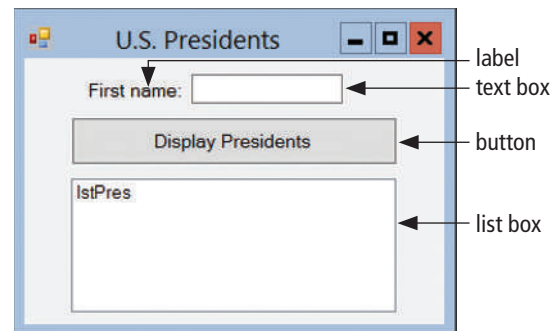


FIGURE 1.5 Controls added to the form.

In order to get the program to perform a task, the programmer has to write instructions called **code**. The code is written into a text-editing window called the **Code Editor**. The code tells the computer what to do after the button is clicked. The **program** includes the form (with its controls) and the code.

Question: What conventions are used to show keystrokes?

Answer: The combination *key1*+*key2* means “hold down *key1* and then press *key2*”. The combination Ctrl+C places selected material into the Clipboard. The combination *key1*/*key2* means “Release *key1* and then press *key2*”. The combination Alt/F opens the File menu on a menu bar.

Question: *What is the difference between Visual Studio and Visual Basic?*

Answer: Visual Studio is an all-encompassing development environment for creating websites and Windows applications. Visual Basic is a programming language that is part of Visual Studio.

Question: *What is the difference between the Express and Professional editions of Visual Studio 2012?*

Answer: The Express edition (officially called *Visual Studio Express 2012 for Windows Desktop*) is free and is packaged with this textbook. The Professional edition has additional capabilities and must be purchased from Microsoft.

Question: *How can the programs for the examples in this textbook be obtained?*

Answer: See the preface for information on how to download the programs from the Pearson website.

Question: *Are there any adjustments that should be made to Windows before using this textbook?*

Answer: Yes. By default, Windows shows only the base names of files. You should configure Windows to display the filename extensions for all known file types. By default, the size of text and images appearing with Windows 7 and 8 is a bit small. We recommend enlarging them to the *Medium – 125 % DPI* setting. (The programs downloaded from the Pearson website were created using that setting.)

The details for both adjustments are presented in Appendix B in the “Configuring the Windows Environment” section.

Question: *Are there any adjustments that should be made to Visual Basic while using this textbook?*

Answer: Yes. Three adjustments are discussed in the textbook. In Section 2.2, a setting is specified that guarantees flexibility in naming, saving, and discarding programs. In Section 2.3, we specify the number of spaces that lines of code will be indented. In Section 3.2, we set some options that affect how rigorous we must be when declaring the data types of variables.

Question: *Where will new programs be saved?*

Answer: Before writing your first program, you should use File Explorer (with Windows 8) or Windows Explorer (with Windows Vista or Windows 7) to create a separate folder to hold your programs. The first time you save a program, you will have to browse to that folder. Subsequent savings will use that folder as the default folder.

1.2 Program Development Cycle

We learned in Section 1.1 that hardware refers to the machinery in a computer system (such as the monitor, keyboard, and CPU) and software refers to a collection of instructions, called a **program**, that directs the hardware. Programs are written to solve problems or perform tasks on a computer. Programmers translate the solutions or tasks into a language the computer can understand. As we write programs, we must keep in mind that the computer will do only what we instruct it to do. Because of this, we must be very careful and thorough when writing our instructions. **Note:** Microsoft Visual Basic refers to a program as a **project**, **application**, or **solution**.

■ Performing a Task on the Computer

The first step in writing instructions to carry out a task is to determine what the **output** should be—that is, exactly what the task should produce. The second step is to identify the data, or **input**, necessary to obtain the output. The last step is to determine how to **process** the input to

obtain the desired output—that is, to determine what formulas or ways of doing things should be used to obtain the output.

This problem-solving approach is the same as that used to solve word problems in an algebra class. For example, consider the following algebra problem:

How fast is a car moving if it travels 50 miles in 2 hours?

The first step is to determine the type of answer requested. The answer should be a number giving the speed in miles per hour (the output). (*Speed* is also called *velocity*.) The information needed to obtain the answer is the distance and time the car has traveled (the input). The formula

$$\text{speed} = \text{distance}/\text{time}$$

is used to process the distance traveled and the time elapsed in order to determine the speed. That is,

$$\begin{aligned}\text{speed} &= 50 \text{ miles}/2 \text{ hours} \\ &= 25 \text{ miles}/\text{hour}\end{aligned}$$

A graphical representation of this problem-solving process is



We determine what we want as output, get the needed input, and process the input to produce the desired output.

In the chapters that follow, we discuss how to write programs to carry out the preceding operations. But first we look at the general process of writing programs.

■ Program Planning

A baking recipe provides a good example of a plan. The ingredients and the amounts are determined by what is to be baked. That is, the *output* determines the *input* and the *processing*. The recipe, or plan, reduces the number of mistakes you might make if you tried to bake with no plan at all. Although it's difficult to imagine an architect building a bridge or a factory without a detailed plan, many programmers (particularly students in their first programming course) try to write programs without first making a careful plan. The more complicated the problem, the more complex the plan may be. You will spend much less time working on a program if you devise a carefully thought out step-by-step plan and test it before actually writing the program.

Many programmers plan their programs using a sequence of steps, referred to as the **Software Development Life Cycle**. The following step-by-step process will enable you to use your time efficiently and help you design error-free programs that produce the desired output.

1. **Analyze:** Define the problem.

Be sure you understand what the program should do—that is, what the output should be. Have a clear idea of what data (or input) are given and the relationship between the input and the desired output.

2. **Design:** Plan the solution to the problem.

Find a logical sequence of precise steps that solve the problem. Such a sequence of steps is called an **algorithm**. Every detail, including obvious steps, should appear in the algorithm. In

the next section, we discuss three popular methods used to develop the logic plan: flowcharts, pseudocode, and top-down charts. These tools help the programmer break a problem into a sequence of small tasks the computer can perform to solve the problem. Planning also involves using representative data to test the logic of the algorithm by hand to ensure that it is correct.

3. Design the interface: Select the objects (text boxes, buttons, etc.).

Determine how the input will be obtained and how the output will be displayed. Then create objects to receive the input and display the output. Also, create appropriate buttons and menus to allow the user to control the program.

4. Code: Translate the algorithm into a programming language.

Coding is the technical word for writing the program. During this stage, the program is written in Visual Basic and entered into the computer. The programmer uses the algorithm devised in Step 2 along with a knowledge of Visual Basic.

5. Test and correct: Locate and remove any errors in the program.

Testing is the process of finding errors in a program. (An error in a program is called a **bug** and testing and correcting is often referred to as **debugging**.) As the program is typed, Visual Basic points out certain kinds of program errors. Other kinds of errors will be detected by Visual Basic when the program is executed; however, many errors due to typing mistakes, flaws in the algorithm, or incorrect use of the Visual Basic language rules can be uncovered and corrected only by careful detective work. An example of such an error would be using addition when multiplication was the proper operation.

6. Complete the documentation: Organize all the material that describes the program.

Documentation is intended to allow another person, or the programmer at a later date, to understand the program. Internal documentation (comments) consists of statements in the program that are not executed but point out the purposes of various parts of the program. Documentation might also consist of a detailed description of what the program does and how to use it (for instance, what type of input is expected). For commercial programs, documentation includes an instruction manual and on-line help. Other types of documentation are the flowchart, pseudocode, and hierarchy chart that were used to construct the program. Although documentation is listed as the last step in the program development cycle, it should take place as the program is being coded.

1.3 Programming Tools

This section discusses some specific algorithms and describes three tools used to convert algorithms into computer programs: flowcharts, pseudocode, and hierarchy charts.

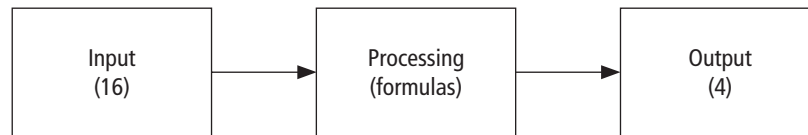
You use algorithms every day to make decisions and perform tasks. For instance, whenever you mail a letter, you must decide how much postage to put on the envelope. One rule of thumb is to use one stamp for every five sheets of paper or fraction thereof. Suppose a friend asks you to determine the number of stamps to place on an envelope. The following algorithm will accomplish the task.

- | | |
|--|--------------|
| 1. Request the number of sheets of paper; call it Sheets. | (input) |
| 2. Divide Sheets by 5. | (processing) |
| 3. Round the quotient up to the next highest whole number; call it Stamps. | (processing) |
| 4. Reply with the number Stamps. | (output) |

The preceding algorithm takes the number of sheets (Sheets) as input, processes the data, and produces the number of stamps needed (Stamps) as output. We can test the algorithm for a letter with 16 sheets of paper.

1. Request the number of sheets of paper; Sheets = 16.
2. Dividing 5 into 16 gives 3.2.
3. Rounding 3.2 up to 4 gives Stamps = 4.
4. Reply with the answer, 4 stamps.

This problem-solving example can be illustrated by



Of the program design tools available, three popular ones are the following:



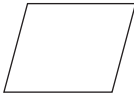

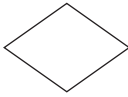

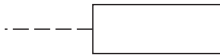
Flowcharts: Graphically depict the logical steps to carry out a task and show how the steps relate to each other.

Pseudocode: Uses English-like phrases with some Visual Basic terms to outline the task.

Hierarchy charts: Show how the different parts of a program relate to each other.

■ Flowcharts

A flowchart consists of special geometric symbols connected by arrows. Within each symbol is a phrase presenting the activity at that step. The shape of the symbol indicates the type of operation that is to occur. For instance, the parallelogram denotes input or output. The arrows connecting the symbols, called **flowlines**, show the progression in which the steps take place. Flowcharts should “flow” from the top of the page to the bottom. Although the symbols used in flowcharts are standardized, no standards exist for the amount of detail required within each symbol.

Symbol	Name	Meaning
	Flowline	Used to connect symbols and indicate the flow of logic.
	Terminal	Used to represent the beginning (Start) or the end (End) of a task.
	Input/Output	Used for input and output operations, such as reading and displaying. The data to be read or displayed are described inside.
	Processing	Used for arithmetic and data-manipulation operations. The instructions are listed inside the symbol.
	Decision	Used for any logic or comparison operations. Unlike the input/output and processing symbols, which have one entry and one exit flowline, the decision symbol has one entry and two exit paths. The path chosen depends on whether the answer to a question is “yes” or “no.”
	Connector	Used to join different flowlines.
	Annotation	Used to provide additional information about another flowchart symbol.

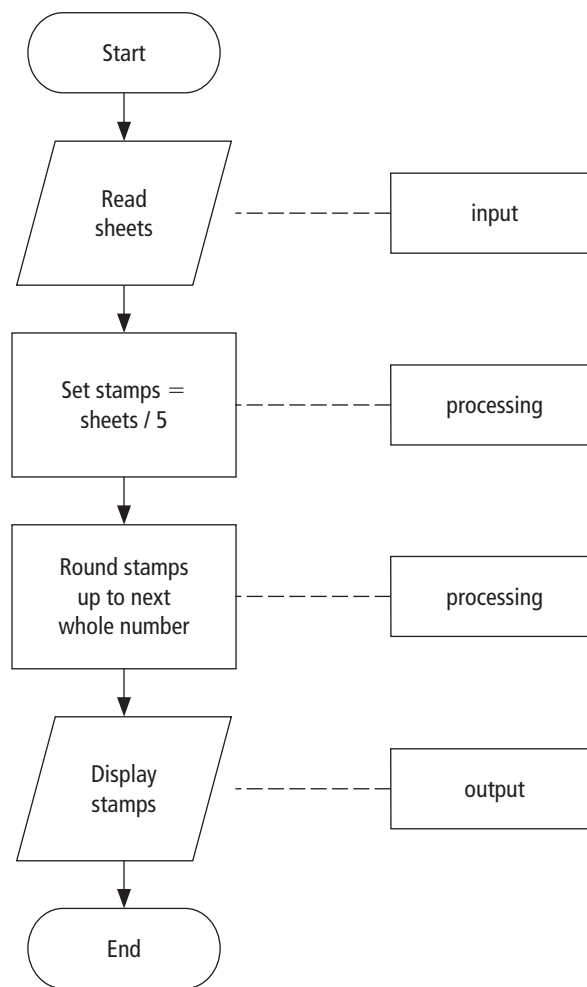


FIGURE 1.6 Flowchart for the postage-stamp problem.

The table of the flowchart symbols shown on the previous page has been adopted by the American National Standards Institute (ANSI). Figure 1.6 shows the flowchart for the postage-stamp problem.

The main advantage of using a flowchart to plan a task is that it provides a graphical representation of the task, which makes the logic easier to follow. We can clearly see every step and how each is connected to the next. The major disadvantage is that when a program is very large, the flowcharts may continue for many pages, making them difficult to follow and modify.

■ Pseudocode

Pseudocode is an abbreviated plain English version of actual computer code (hence, *pseudocode*). The geometric symbols used in flowcharts are replaced by English-like statements that outline the process. As a result, pseudocode looks more like computer code than does a flowchart. Pseudocode allows the programmer to focus on the steps required to solve a problem rather than on how to use the computer language. The programmer can describe the algorithm in Visual Basic–like form without being restricted by the rules of Visual Basic. When the pseudocode is completed, it can be easily translated into the Visual Basic language.

The following is pseudocode for the postage-stamp problem:

Program: Determine the proper number of stamps for a letter.

Read Sheets

(input)

Set the number of stamps to $\text{Sheets} / 5$

(processing)

Round the number of stamps up to the next whole number

(processing)

Display the number of stamps

(output)

Pseudocode has several advantages. It is compact and probably will not extend for many pages as flowcharts commonly do. Also, the plan looks like the code to be written and so is preferred by many programmers.

■ Hierarchy Chart

The last programming tool we'll discuss is the **hierarchy chart**, which shows the overall program structure. Hierarchy charts are also called structure charts, HIPO (Hierarchy plus Input-Process-Output) charts, top-down charts, or VTOC (Visual Table of Contents) charts. All these names refer to planning diagrams that are similar to a company's organization chart.

Hierarchy charts depict the organization of a program but omit the specific processing logic. They describe what each part, or **module**, of the program does and they show how the modules relate to each other. The details on how the modules work, however, are omitted. The chart is read from top to bottom and from left to right. Each module may be subdivided into a succession of submodules that branch out under it. Typically, after the activities in the succession of submodules are carried out, the module to the right of the original module is considered. A quick glance at the hierarchy chart reveals each task performed in the program and where it is performed. Figure 1.7 shows a hierarchy chart for the postage-stamp problem.

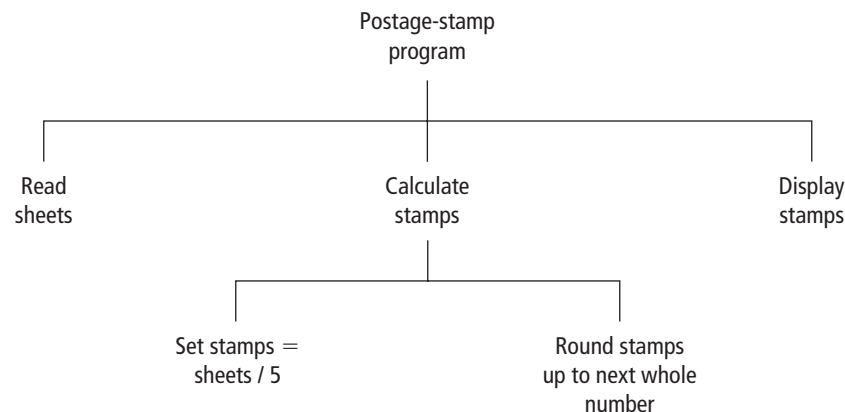


FIGURE 1.7 Hierarchy chart for the postage-stamp problem.

The main benefit of hierarchy charts is in the initial planning of a program. We break down the major parts of a program so we can see what must be done in general. From this point, we can then refine each module into more detailed plans using flowcharts or pseudocode. This process is called the **divide-and-conquer** method.

■ Decision Structure

The postage-stamp problem was solved by a series of instructions to read data, perform calculations, and display results. Each step was in a sequence; that is, we moved from one line to the next without skipping over any lines. This kind of structure is called a **sequence structure**. Many problems, however, require a decision to determine whether a series of instructions should be executed. If the answer to a question is “yes”, then one group of instructions is executed. If the

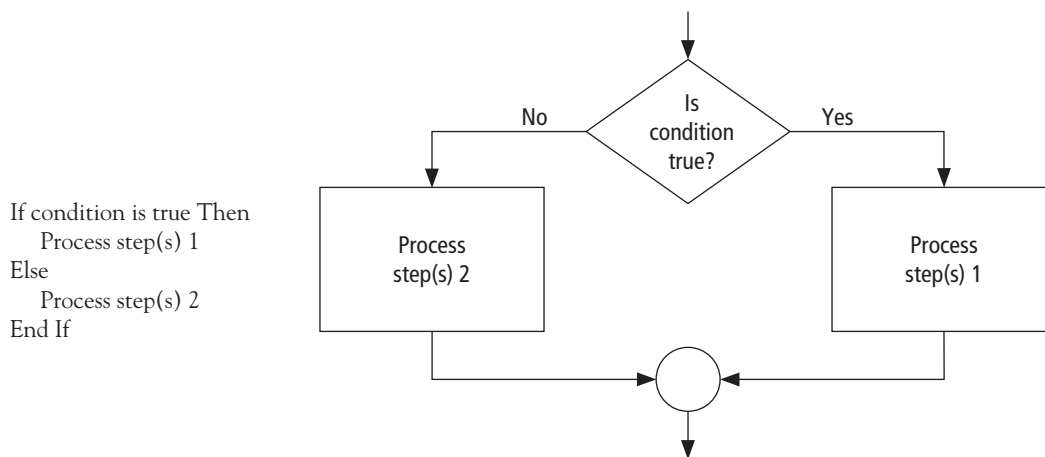


FIGURE 1.8 Pseudocode and flowchart for a decision structure.

answer is “no”, then another is executed. This structure is called a **decision structure**. Figure 1.8 contains the pseudocode and flowchart for a decision structure.

Sequence and decision structures are both used to solve the following problem.

■ Direction of Numbered NYC Streets Algorithm

Problem: Given a street number of a one-way street in New York City, decide the direction of the street, either eastbound or westbound.

Discussion: There is a simple rule to tell the direction of a one-way street in New York City: Even-numbered streets run eastbound.

Input: Street number.

Processing: Decide if the street number is divisible by 2.

Output: “Eastbound” or “Westbound”.

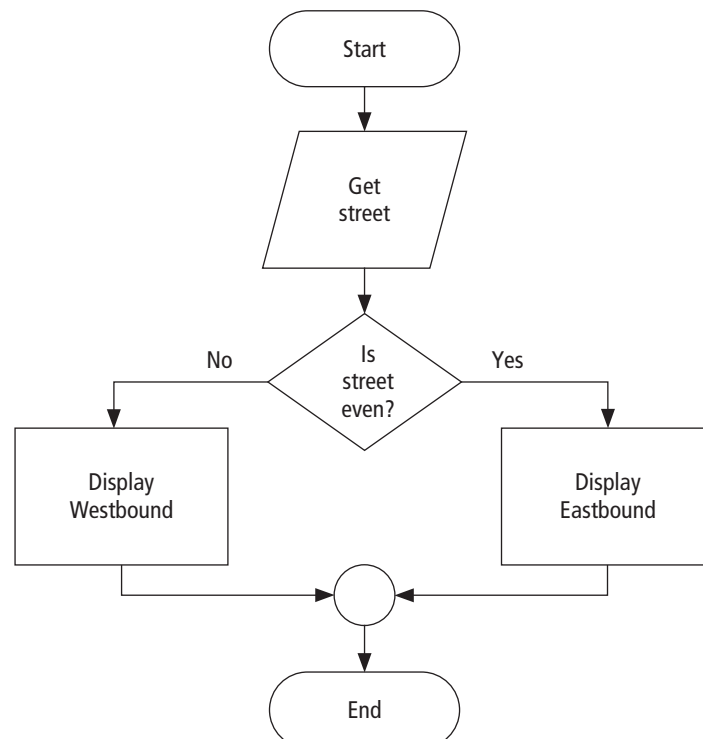


FIGURE 1.9 Flowchart for the numbered New York City streets problem.

Program: Determine the direction of a numbered NYC street.
 Get street
 If street is even Then
 Display Eastbound
 Else
 Display Westbound
 End If

FIGURE 1.10 Pseudocode for the numbered New York City streets problem.

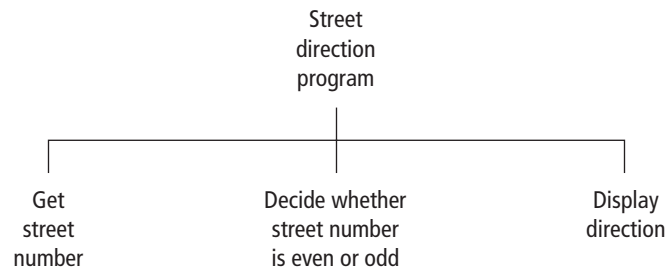


FIGURE 1.11 Hierarchy chart for the numbered New York City streets problem.

Figures 1.9 through 1.11 show the flowchart, pseudocode, and hierarchy chart for the numbered New York City streets problem.

■ Repetition Structure

A programming structure that executes instructions many times is called a **repetition structure** or a **loop structure**. Loop structures need a test (or condition) to tell when the loop should end. Without an exit condition, the loop would repeat endlessly (an infinite loop). One way to control the number of times a loop repeats (often referred to as the number of passes or iterations) is to check a condition before each pass through the loop and continue executing the loop as long as the condition is true. See Fig. 1.12. The solution of the next problem requires a repetition structure.

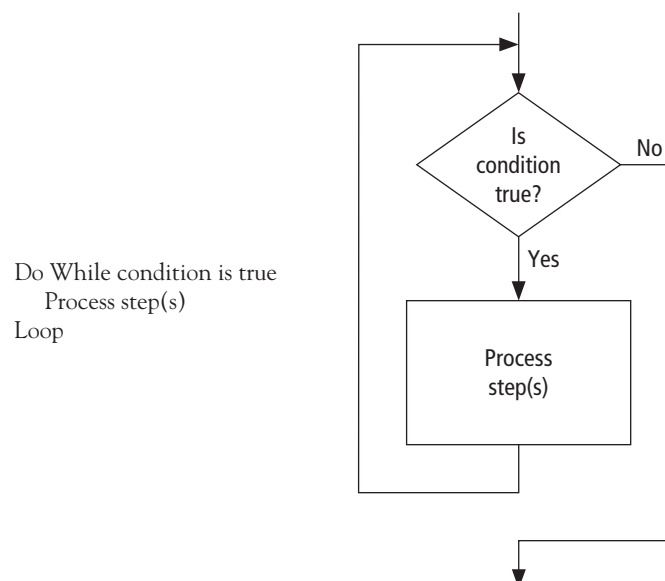


FIGURE 1.12 Pseudocode and flowchart for a loop.

■ Class Average Algorithm

Problem: Calculate and report the average grade for a class.

Discussion: The average grade equals the sum of all grades divided by the number of students. We need a loop to read and then add (accumulate) the grades for each student in the class. Inside the loop, we also need to total (count) the number of students in the class. See Figs. 1.13 to 1.15.

Input: Student grades.

Processing: Find the sum of the grades; count the number of students; calculate average grade = sum of grades / number of students.

Output: Average grade.

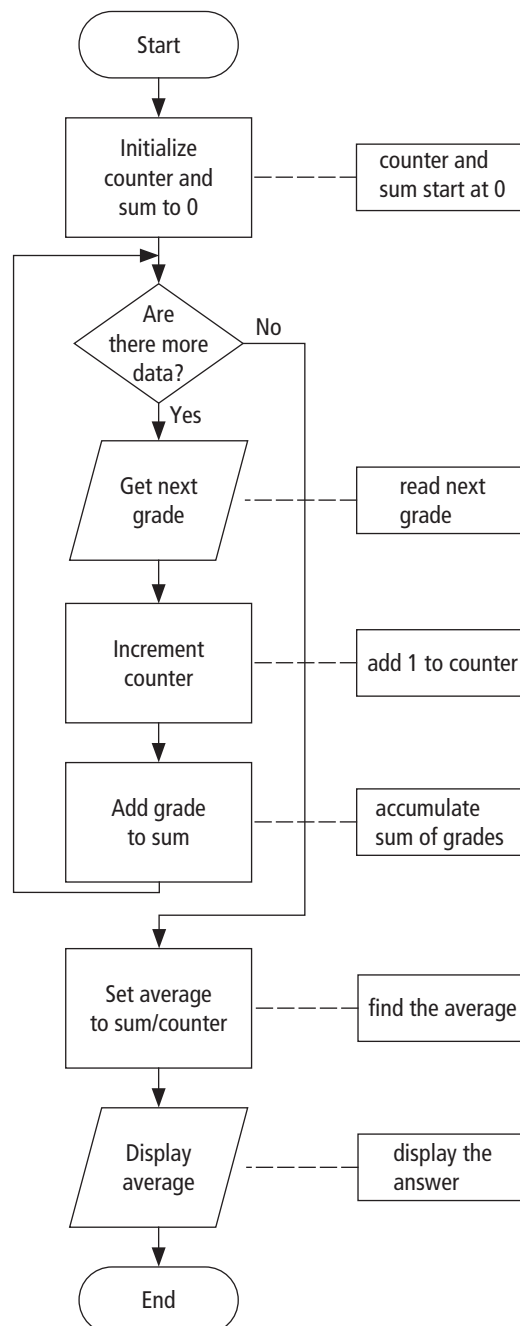


FIGURE 1.13 Flowchart for the class average problem.

Program: Calculate and report the average grade of a class.
 Initialize Counter and Sum to 0
 Do While there are more data
 Get the next Grade
 Increment the Counter
 Add the Grade to the Sum
 Loop
 Compute Average = Sum/Counter
 Display Average

FIGURE 1.14 Pseudocode for the class average problem.

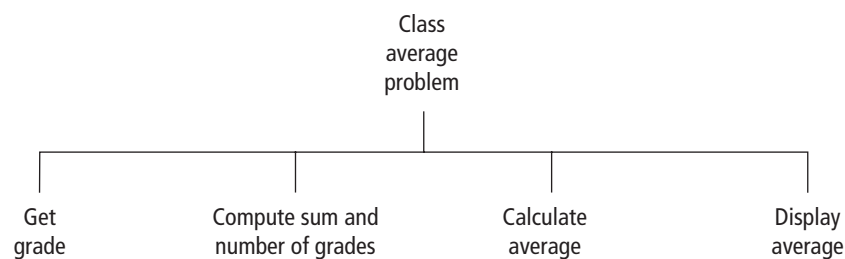


FIGURE 1.15 Hierarchy chart for the class average problem.

■ Comments

1. Tracing a flowchart is like playing a board game. We begin at the Start symbol and proceed from symbol to symbol until we reach the End symbol. At any time, we will be at just one symbol. In a board game, the path taken depends on the result of spinning a spinner or rolling a pair of dice. The path taken through a flowchart depends on the input.
2. The algorithm should be tested at the flowchart stage before being coded into a program. Different data should be used as input, and the output checked. This process is known as **desk checking**. The test data should include nonstandard data as well as typical data.
3. Flowcharts, pseudocode, and hierarchy charts are universal problem-solving tools. They can be used to plan programs for implementation in many computer languages, not just Visual Basic.
4. Flowcharts are used throughout this text to provide a visualization of the flow of certain programming tasks and Visual Basic control structures. Major examples of pseudocode and hierarchy charts appear in the case studies.
5. Flowcharts are time-consuming to write and difficult to update. For this reason, professional programmers are more likely to favor pseudocode and hierarchy charts. Because flowcharts so clearly illustrate the logical flow of programming techniques, however, they are a valuable tool in the education of programmers.
6. There are many styles of pseudocode. Some programmers use an outline form, whereas others use a form that looks almost like a programming language. The pseudocode appearing in the case studies of this text focuses on the primary tasks to be performed by the program and leaves many of the routine details to be completed during the coding process. Several Visual Basic keywords, such as “If”, “Else”, “Do”, and “While”, are used extensively in the pseudocode appearing in this text.

2

Visual Basic, Controls, and Events



2.1 An Introduction to Visual Basic 2012 38

- ◆ Why Windows and Why Visual Basic? ◆ How You Develop a Visual Basic Program
- ◆ The Different Versions of Visual Basic

2.2 Visual Basic Controls 40

- ◆ Starting a New Visual Basic Program ◆ A Text Box Walkthrough
- ◆ A Button Walkthrough ◆ A Label Walkthrough ◆ A List Box Walkthrough
- ◆ The Name Property ◆ Fonts ◆ Auto Hide ◆ Positioning and Aligning Controls
- ◆ Multiple Controls ◆ Setting Tab Order

2.3 Visual Basic Events 58

- ◆ An Event Procedure Walkthrough ◆ Properties and Event Procedures of the Form
- ◆ The Header of an Event Procedure ◆ Opening a Program

Summary 74

2.1 An Introduction to Visual Basic 2012

Visual Basic 2012 is the latest generation of Visual Basic, a language used by many software developers. Visual Basic was designed to make user-friendly programs easier to develop. Prior to the creation of Visual Basic, developing a friendly user interface usually required a programmer to use a language such as C or C++, often requiring hundreds of lines of code just to get a window to appear on the screen. Now the same program can be created in much less time with fewer instructions.

■ Why Windows and Why Visual Basic?

What people call **graphical user interfaces**, or GUIs (pronounced “gooies”), have revolutionized the software industry. Instead of the confusing textual prompts that earlier users once saw, today’s users are presented with such devices as icons, buttons, and drop-down lists that respond to mouse clicks. Accompanying the revolution in how programs look was a revolution in how they feel. Consider a program that requests information for a database. Figure 2.1 shows how a program written before the advent of GUIs got its information. The program requests the six pieces of data one at a time, with no opportunity to go back and alter previously entered information. Then the screen clears and the six inputs are again requested one at a time.

Enter name (Enter EOD to terminate): Mr. President
Enter Address: 1600 Pennsylvania Avenue
Enter City: Washington
Enter State: DC
Enter Zip code: 20500
Enter Phone Number: 202-456-1414

FIGURE 2.1 Input screen of a pre-Visual Basic program to fill a database.

Figure 2.2 shows how an equivalent Visual Basic program gets its information. The boxes may be filled in any order. When the user clicks on a box with the mouse, the cursor moves to that box. The user can either type in new information or edit the existing information. When satisfied that all the information is correct, the user clicks on the *Write to Database* button. The boxes will clear, and the data for another person can be entered. After all names have been entered, the user clicks on the *Exit* button. In Fig. 2.1, the program is in control; in Fig. 2.2, the user is in control!

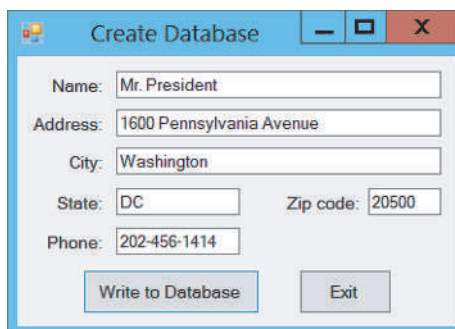


FIGURE 2.2 Input screen of a Visual Basic program to fill a database.

■ How You Develop a Visual Basic Program

A key element of planning a Visual Basic program is deciding what the user sees—in other words, designing the user interface. What data will he or she be entering? How large a window should the program use? Where will you place the buttons the user clicks on to activate actions in the program? Will the program have places to enter text (text boxes) and places to display output? What kind of warning boxes (message boxes) should the program use? In Visual Basic, the responsive objects a program designer places on windows are called *controls*. Two features make Visual Basic different from traditional programming tools:

1. You literally draw the user interface, much like using a paint program.
2. Perhaps more important, when you're done drawing the interface, the buttons, text boxes, and other objects that you have placed in a blank window will automatically recognize user actions such as mouse movements and button clicks. That is, the sequence of procedures executed in your program is controlled by “events” that the user initiates rather than by a predetermined sequence of procedures in your program.

In any case, only after you design the interface does anything like traditional programming occur. Objects in Visual Basic recognize events like mouse clicks; how the objects respond to them depends on the instructions you write. You always need to write instructions in order to make controls respond to events. This makes Visual Basic programming fundamentally different from traditional programming. Programs in traditional programming languages ran from the top down. For these programming languages, execution started from the first line and moved with the flow of the program to different parts as needed. A Visual Basic program works differently. Its core is a set of independent groups of instructions that are activated by the events they have been told to recognize. This event-driven methodology is a fundamental shift. The user decides the order in which things happen, not the programmer.

Most of the programming instructions in Visual Basic that tell your program how to respond to events like mouse clicks occur in what Visual Basic calls *event procedures*. Essentially, anything executable in a Visual Basic program either is in an event procedure or is used by an event procedure to help the procedure carry out its job. In fact, to stress that Visual Basic is fundamentally different from traditional programming languages, Microsoft uses the term *project* or *application*, rather than *program*, to refer to the combination of programming instructions and user interface that makes a Visual Basic program possible. Here is a summary of the steps you take to design a Visual Basic program:

1. Design the appearance of the window that the user sees.
2. Determine the events that the controls on the window should respond to.
3. Write the event procedures for those events.

Now here is what happens when the program is running:

1. Visual Basic monitors the controls in the window to detect any event that a control can recognize (mouse movements, clicks, keystrokes, and so on).
2. When Visual Basic detects an event, it examines the program to see if you've written an event procedure for that event.
3. If you have written an event procedure, Visual Basic executes the instructions that make up that event procedure and goes back to Step 1.
4. If you have not written an event procedure, Visual Basic ignores the event and goes back to Step 1.

These steps cycle continuously until the program ends. Usually, an event must happen before Visual Basic will do anything. Event-driven programs are more reactive than active—and that makes them more user friendly.

■ The Different Versions of Visual Basic

Visual Basic 1.0 first appeared in 1991. It was followed by version 2.0 in 1992, version 3.0 in 1993, version 4.0 in 1995, version 5.0 in 1997, and version 6.0 in 1998. VB.NET, initially released in February 2002, was not backward compatible with the earlier versions of Visual Basic. It incorporated many features requested by software developers, such as true inheritance. Visual Basic 2005, released in November 2005, Visual Basic 2008, released in November 2007, Visual Basic 2010, released in April 2010, and Visual Basic 2012, released in October 2012 are significantly improved versions of VB.NET.

2.2 Visual Basic Controls

Visual Basic programs display a Windows-style screen (called a **form**) with boxes into which users type (and in which users edit) information and buttons that they click on to initiate actions. The boxes and buttons are referred to as **controls**. In this section, we examine forms and four of the most useful Visual Basic controls.

■ Starting a New Visual Basic Program

Each program is saved (as several files and subfolders) in its own folder. Before writing your first program, you should use File Explorer (with Windows 8) or Windows Explorer (with Windows Vista or Windows 7) to create a folder to hold your programs.

The process for starting Visual Basic varies slightly with the version of Windows and the edition of Visual Studio installed on the computer. Some possible sequences of steps are shown below.

Windows Vista or Windows 7 Click the Windows *Start* button, click *All Programs*, and then click on “Microsoft Visual Studio 2012 Express for Desktop.”

Windows 8 Click the tile labeled “VS Express for Desktop.” If there is no such tile, Click on *Search* in the Charms bar, select the Apps category, type “VS” into the Search box in the upper-right part of the screen, and click on the rectangle labeled “VS Express for Desktop” that appears on the left side of the screen.

Figure 2.3 shows the top part of the screen after Visual Basic is started. A Menu bar and a Toolbar are at the top of the screen. These two bars, with minor variations, are always present while you are working with Visual Basic. The remainder of the screen is called the Start Page. Some tasks can be initiated from the Menu bar, the Toolbar, and the Start Page. We will usually initiate them from the Menu bar or the Toolbar.

The first item on the Menu bar is FILE. Click on FILE, and then click on *New Project* to produce a New Project dialog box. Figure 2.4 shows a New Project dialog box produced by Visual Basic Express. Your screen might look somewhat different than Fig. 2.4 even if you are using the Express edition of Visual Studio.

Select *Visual Basic* in the *Templates* list on the left side of Fig. 2.4, and select *Windows Forms Application* in the center list. **Note:** The number of items in the center list will vary depending on the edition of Visual Studio you are using.

The name of the program, initially set to `WindowsApplication1`, can be specified at this time. Since we will have a chance to change it later, let’s just use the name `WindowsApplication1` for

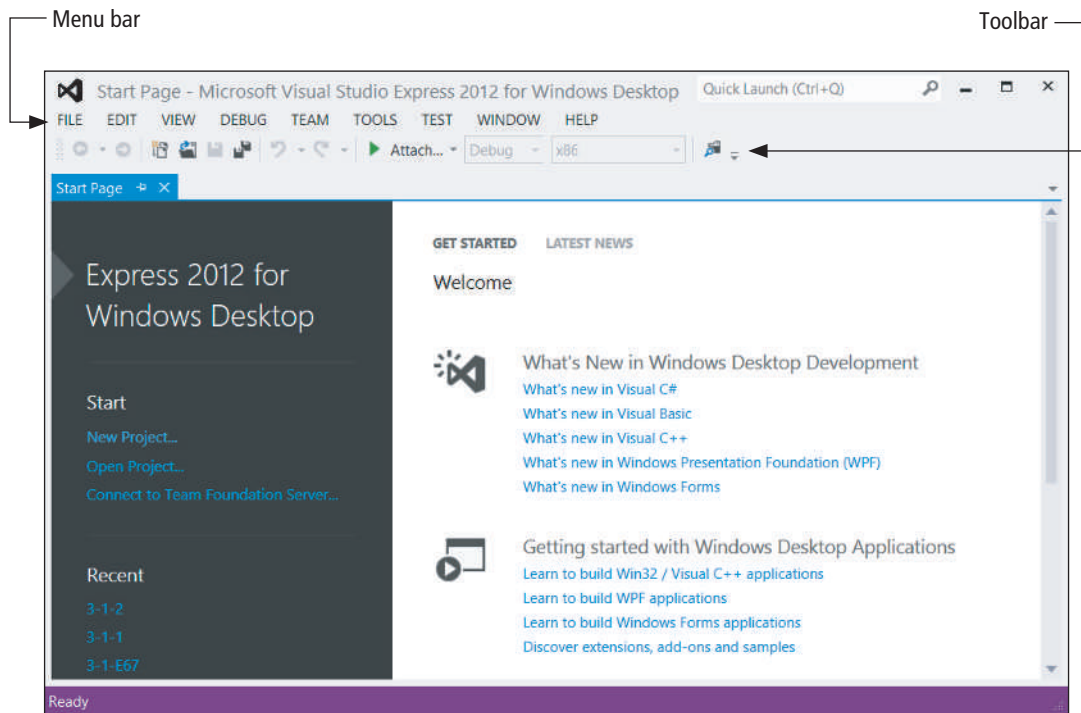


FIGURE 2.3 Visual Basic opening screen.

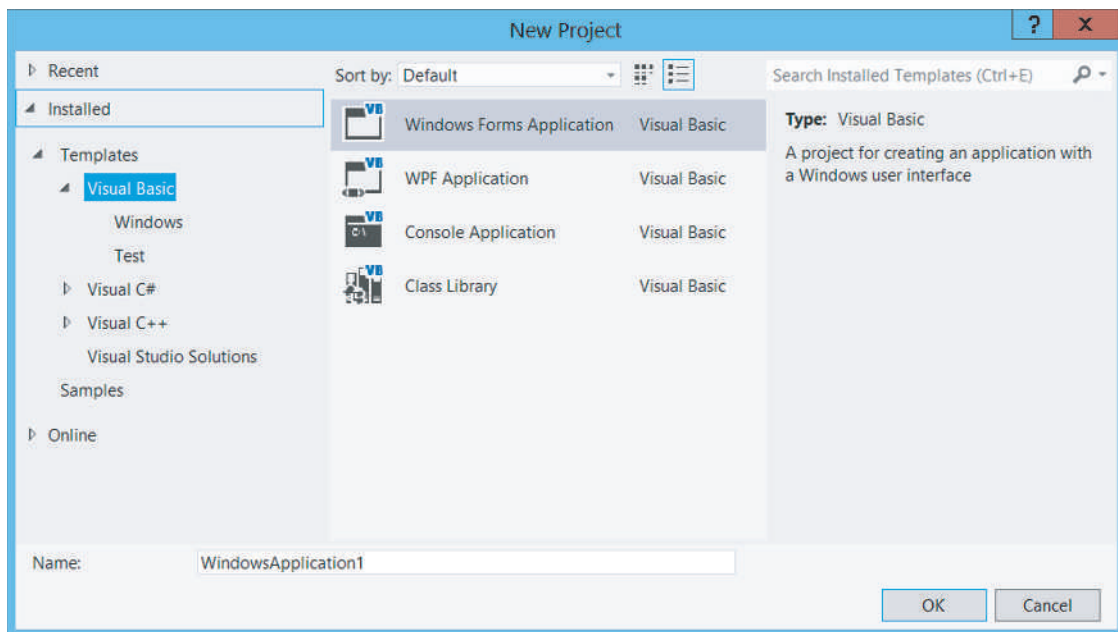


FIGURE 2.4 The Visual Basic New Project dialog box.

now. Click on the OK button to invoke the Visual Basic programming environment. See Fig. 2.5 on the next page. The Visual Basic programming environment is referred to as the **Integrated Development Environment** or **IDE**.

It is possible that your screen will look different than Fig. 2.5. The IDE is extremely configurable. Each window in Fig. 2.5 can have its location and size altered. New windows can be

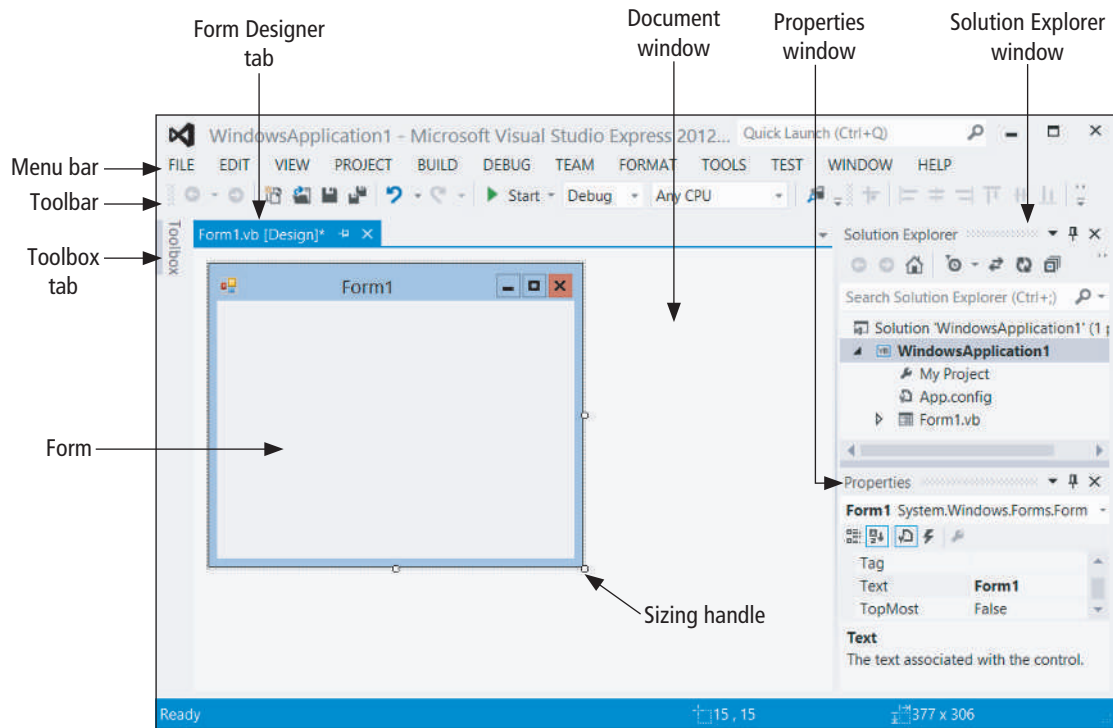


FIGURE 2.5 The Visual Basic Integrated Development Environment in Form Designer mode.

displayed in the IDE, and any window can be closed or hidden behind a tab. For instance, in Fig. 2.5 the Toolbox window is hidden behind a tab. The **VIEW** menu is used to add additional windows to the IDE. If you would like your screen to look similar to Fig. 2.5, click on *Reset Windows Layout* in the **WINDOW** menu, and then click on the **Yes** button.

The **Menu bar** of the IDE displays the menus of commands you use to work with Visual Basic. Some of the menus, like **FILE**, **EDIT**, **VIEW**, and **WINDOW**, are common to most Windows applications. Others, such as **PROJECT**, **DEBUG**, and **DATA**, provide commands specific to programming in Visual Basic.

The **Toolbar** holds a collection of buttons that carry out standard operations when clicked. For example, you use the sixth button, which looks like two diskettes, to save the files associated with the current program. To reveal the purpose of a Toolbar button, hover the mouse pointer over it. The little information rectangle that pops up is called a **tooltip**.

The **Document window** currently holds the rectangular **Form window**, or **form** for short. The form becomes a Windows window when a program is executed. Most information displayed by the program appears on the form. The information usually is displayed in controls that the programmer has placed on the form. **Note:** You can change the size of the form by dragging one of its sizing handles.

The **Properties window** is used to change the appearance and behavior of objects on the form.

The **Solution Explorer** window displays the files associated with the program and provides access to the commands that pertain to them. (**Note:** If the Solution Explorer or the Properties window is not visible, click on it in the **VIEW** menu.)

The **Toolbox** holds icons representing objects (called controls) that can be placed on the form. If your screen does not show the Toolbox, hover the mouse over the Toolbox tab at the left side of the screen. The Toolbox will slide into view. Then click on the pushpin icon in the title bar at the top of the Toolbox to keep the Toolbox permanently displayed in the IDE. (**Note:** If there is no tab marked **Toolbox**, click on *Toolbox* in the **VIEW** menu.)

The controls in the Toolbox are grouped into categories such as *All Windows Forms* and *Common Controls*. Figure 2.6 shows the Toolbox after the *Common Controls* group has been expanded. Most of the controls discussed in this text can be found in the list of common controls. (You can obtain a description of a control by hovering the mouse over the control.) The four controls discussed in this chapter are text boxes, labels, buttons, and list boxes. In order to see all the group names, collapse each of the groups.

Text boxes: Text boxes are used to get information from the user, referred to as **input**, or to display information produced by the program, referred to as **output**.

Labels: Labels are placed near text boxes to tell the user what type of information is displayed in the text boxes.

Buttons: The user clicks on a button to initiate an action.

List boxes: In the first part of this book, list boxes are used to display output. Later, they are used to make selections.



VideoNote
Visual Basic
Controls
and Events

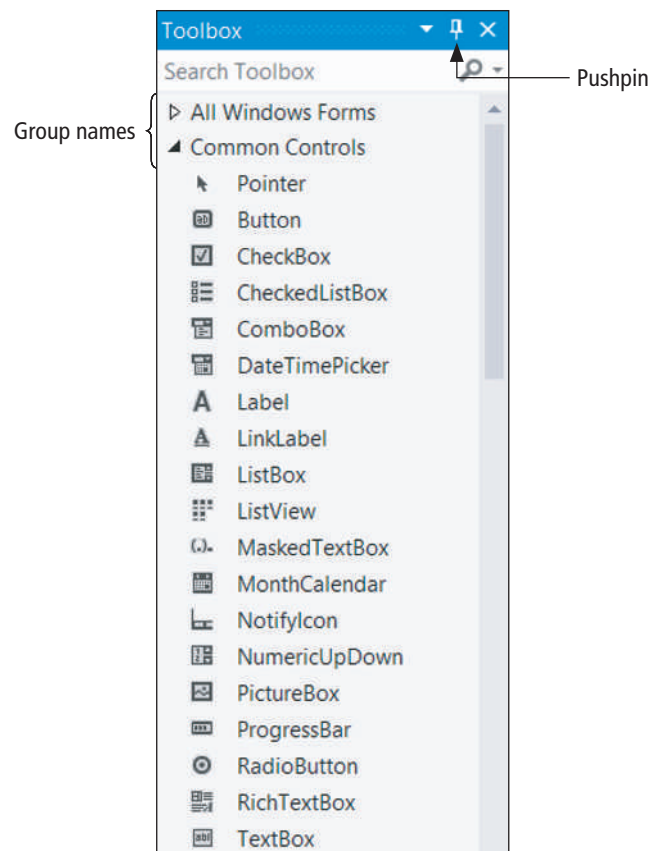


FIGURE 2.6 The Toolbox's common controls.

■ An Important Setting


The process of naming and saving programs can proceed in two different ways. In this book, we do not require that a program be given a name until it is saved. The following steps guarantee that Visual Basic will follow that practice.

1. Click on *Options* from the **TOOLS** menu to display an Options dialog box.
2. Click on the *Projects and Solutions* item in the left pane of the Options dialog box.

3. If the box labeled “Save new projects when created” is checked, uncheck it.
4. Click on the OK button.
5. Open the FILE menu in the Toolbar and click on *Close Solution* (or *Close Project*). **Note:** If a dialog box appears and asks you if you want to save or discard changes to the current project, click on the *Discard* button.

■ A Text Box Walkthrough

Place a text box on a form

1. Start a new Visual Basic program.
2. Double-click on the TextBox control ( **TextBox**) in the *Common Controls* group of the Toolbox.

A rectangle with three small squares appears at the upper-left corner of the form. The square on the top of the text box, called the **Tasks button**, can be used to set the MultiLine property of the text box. The squares on the left and right sides of the text box are called **sizing handles**. See Fig. 2.7. An object showing its handles is said to be **selected**. A selected text box can have its width altered, location changed, and other properties modified. You alter the width of the text box by dragging one of its sizing handles.

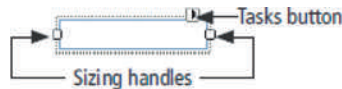


FIGURE 2.7 Setting the Text property.

3. Move the mouse cursor to any point in the interior of the text box, hold down the left mouse button, and drag the text box to the center of the form.
4. Click anywhere on the form outside the rectangle to deselect the text box.
5. Click on the rectangle to reselect the text box.
6. Hover the mouse over the handle in the center of the right side of the text box until the cursor becomes a double-arrow, hold down the left mouse button, and move the mouse to the right.

The text box is stretched to the right. Similarly, grabbing the handle on the left side and moving the mouse to the left stretches the text box to the left. You also can use the handles to make the text box smaller. Steps 2, 3, and 6 allow you to place a text box of any width anywhere on the form. **Note:** The text box should now be selected; that is, its sizing handles should be showing. If not, click anywhere inside the text box to select it.

7. Press the Delete key to remove the text box from the form.

Step 8 gives an alternative way to place a text box of any width at any location on the form.




8. Click on the text box icon in the Toolbox, move the mouse pointer to any place on the form, hold down the left mouse button, drag the mouse on a diagonal, and release the mouse button to create a selected text box.

You can now alter the width and location as before. **Note:** The text box should now be selected. If not, click anywhere inside the text box to select it.

Activate, move, and resize the Properties window

9. Press F4 to activate the Properties window.

You also can activate the Properties window by clicking on it, clicking on *Properties Window* from the VIEW menu, or right-clicking on the text box with the mouse button and selecting

Properties from the context menu that appears. See Fig. 2.8. The first line of the Properties window (called the **Object box**) reads “TextBox1”, etc. TextBox1 is the current name of the text box. The third button in the row of buttons below the Object box, the *Properties* button () is normally highlighted. If not, click on it. The left column of the Properties window gives the available properties, and the right column gives the current settings of the properties. The first two buttons ( ) in the row of buttons below the Object box permit you to view the list of properties either grouped into categories or alphabetically. You can use the up- and down-arrow keys (or the scroll arrows, scroll box, or the mouse scroll wheel) to move through the list of properties.

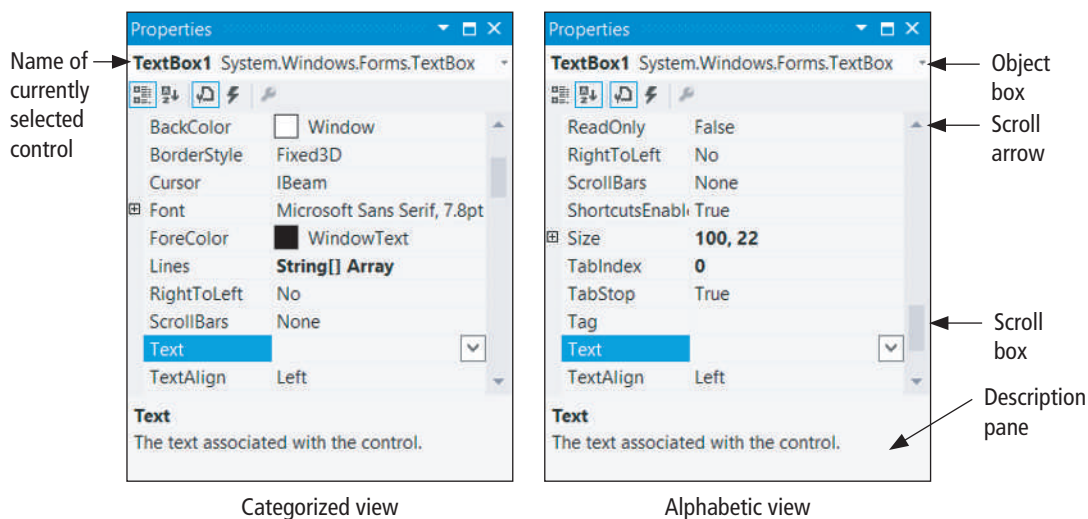


FIGURE 2.8 Text box Properties window.

10. Click on the Properties window's title bar and drag the window to the center of the screen.
The Properties window is said to be **floating** or **undocked**. Some people find a floating window easier to work with.
11. Drag the lower-right corner of the Properties window to change the size of the Properties window.
An enlarged window will show more properties at once.
12. Hold down the Ctrl key and double-click on the title bar.
The Properties window will return to its original docked location. We now will discuss four properties in this walkthrough.

Set four properties of the text box

Assume that the text box is selected and its Properties window activated.

Note 1: The third and fourth buttons below the Object box, the *Properties* button and the *Events* button, determine whether properties or events are displayed in the Properties window. Normally the *Properties* button is highlighted. If not, click on it.

Note 2: If the Description pane is not visible, right-click on the Properties window, then click on *Description*. The Description pane describes the currently highlighted property.

13. Move to the Text property with the up- and down-arrow keys (alternatively, scroll until the Text property is visible, and click on the property).
The Text property, which determines the words displayed in the text box, is now highlighted. Currently, there is no text displayed in the Text property's Settings box on its right.

14. Type your first name, and then press the Enter key or click on another property. Your name now appears in both the Settings box and the text box. See Fig. 2.9.

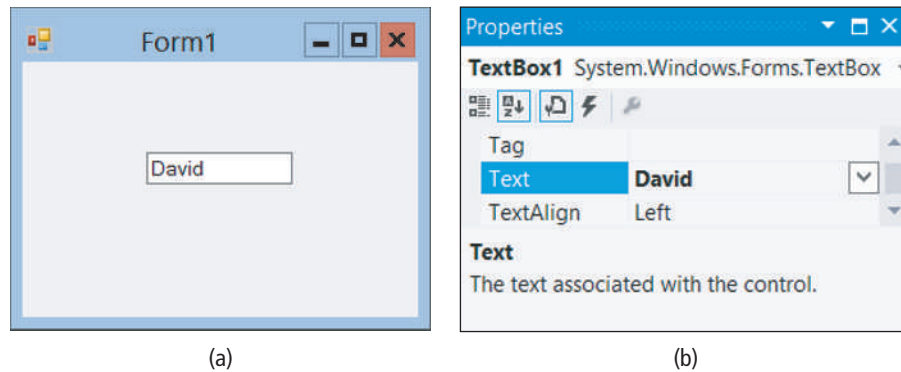


FIGURE 2.9 Setting the Text property.

15. Click at the beginning of your name in the Text Settings box, and add your title, such as Mr., Ms., or The Honorable. Then, press the Enter key.
If you mistyped your name, you can easily correct it now.
16. Use the mouse scroll wheel to move to the ForeColor property, and then click on it.
The ForeColor property determines the color of the text displayed in the text box.
17. Click on the down-arrow button (▼) in the right part of the Settings box, and then click on the Custom tab to display a selection of colors. See Fig. 2.10.

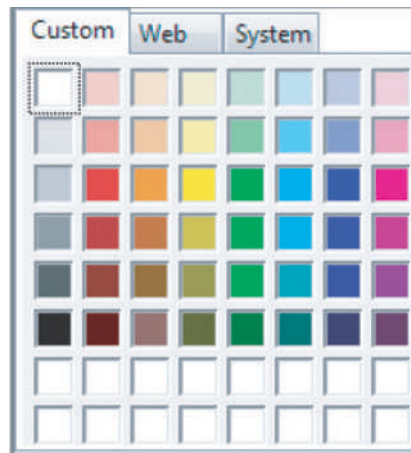


FIGURE 2.10 Setting the ForeColor property.

18. Click on one of the colors, such as *blue* or *red*.
Notice the change in the color of your name.
19. Select the Font property with a single click of the mouse, and click on the ellipsis button (⋮) in the right part of its Settings box.
The Font dialog box in Fig. 2.11 is displayed. The three lists give the current name (Microsoft Sans Serif), current style (Regular), and current size (8 point) of the font. You can change any of these attributes by clicking on an item in its list or by typing into the box at the top of the list.

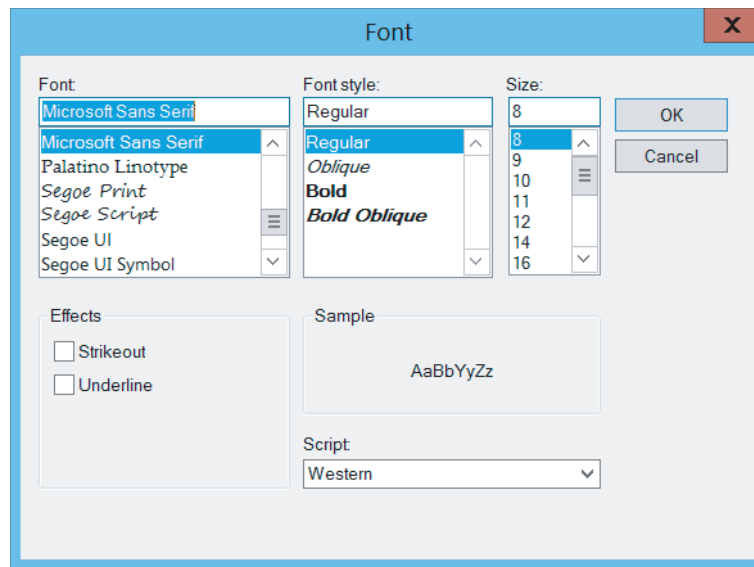


FIGURE 2.11 The Font dialog box.

20. Click on **Bold** in the *Font style* list, click on **12** in the *Size* list, and click on the **OK** button. Your name is now displayed in a larger bold font. The text box will expand so that it can accommodate the larger font.

21. Click on the text box and resize it to be about 3 inches wide.

Visual Basic programs consist of three parts: interface, values of properties, and code. Our interface consists of a form with a single object—a text box. We have set a few properties for the text box—the text (namely, your name), the foreground color, the font style, and the font size. In Section 2.3, we discuss how to place code into a program. Visual Basic endows certain capabilities to programs that are independent of any code we write. We will now run the current program without adding any code and experience these capabilities.

Run and end the program

22. Click on the *Start* button (▶) on the Toolbar to run the program.

Alternatively, you can press F5 to run the program or can click on *Start Debugging* in the **DEBUG** menu. After a brief delay, a copy of the form appears with your name highlighted.

23. Press the **End** key to move the cursor to the end of your name, type in your last name, and then keep typing.

Eventually, the words will scroll to the left.

24. Press the **Home** key to return to the beginning of your name.

The text box functions like a miniature word processor. You can place the cursor anywhere you like in order to add or delete text. You can drag the cursor across text to select a block, place a copy of the block in the Clipboard with **Ctrl+C**, and then duplicate it elsewhere with **Ctrl+V**.

25. Click on the *Stop Debugging* button (■) on the Toolbar to end the program.

Alternately, you can end the program by clicking on the form's *Close* button (✕), clicking on *Stop Debugging* in the **DEBUG** menu, or pressing **Alt+F4**.

26. Select the text box, activate the Properties window, select the **ReadOnly** property, click on the down-arrow button (▼), and finally click on **True**.

Notice that the background color of the text box has turned gray.

27. Run the program, and try typing into the text box. You can't.

Such a text box is used for output. Only code can display information in the text box. (**Note:** In this textbook, whenever a text box will be used only for the purpose of displaying output, we will always set the `ReadOnly` property to `True`.)

28. End the program.

Saving and closing the program

29. Click on the Toolbar's *Save All* button () to save the work done so far.

Alternatively, you can click on *Save All* in the FILE menu. The dialog box in Fig. 2.12 will appear to request a name and the location where the program is to be saved.

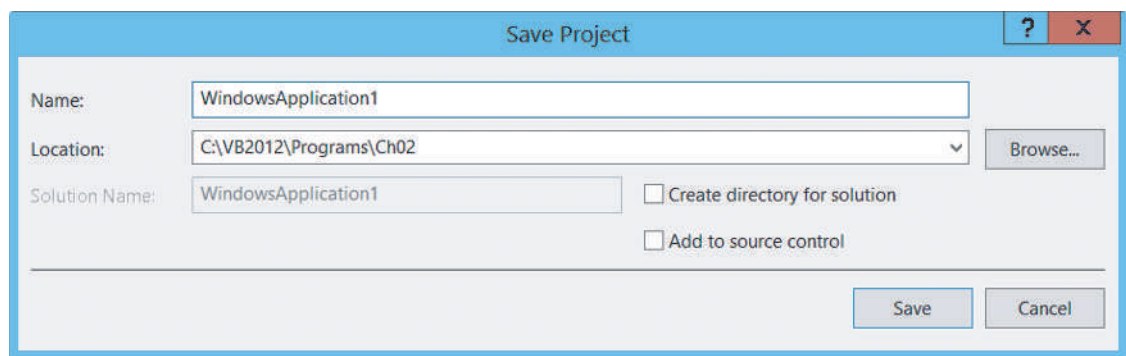


FIGURE 2.12 The Save Project dialog box.

30. Type a name for the program, such as “VBdemo”.

Use Browse to locate a folder. (This folder will automatically be used the next time you click on the *Save All* button.) The files for the program will be saved in a subfolder of the selected folder.

Important: If the “Create directory for solution” check box is checked, then click on the check box to uncheck it.

31. Click on the *Save* button.
32. Click on *Close Solution* (or *Close Project*) in the FILE menu.

In the next step we reload the program.



33. Click on *Open Project* in the FILE menu, navigate to the folder corresponding to the program you just saved, double-click on its folder, and double-click on the file with extension *sln*.

If you do not see the Form Designer for the program, double-click on `Form1.vb` in the Solution Explorer. The program now exists just as it did after Step 28. You can now modify the program and/or run it.

34. Click on *Close Solution* (or *Close Project*) in the FILE menu to close the program.

■ A Button Walkthrough

Place a button on a form

1. Click on the *New Project* button () on the Toolbar.
2. Double-click on the Button control ( Button) in the Toolbox to place a button on the form. The Button control is the second item in the *Common Controls* group of the Toolbox.

3. Drag the button to the center of the form.
4. Activate the Properties window, highlight the Text property, type “Please Push Me”, and press the Enter key.

The button is too small to accommodate the phrase. See Fig. 2.13.

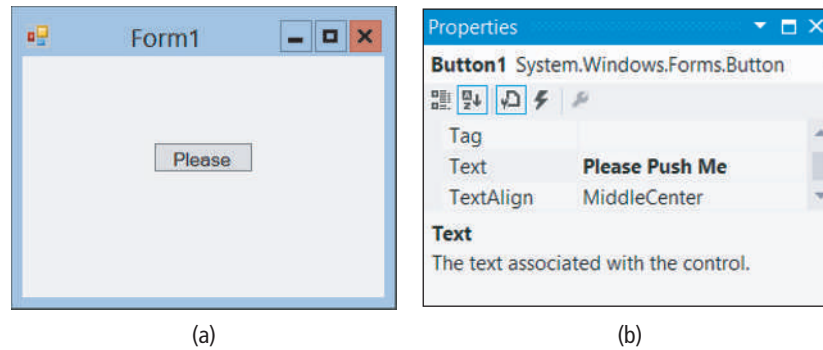


FIGURE 2.13 Setting the Text property.

5. Click on the button to select it, and then drag the right-hand sizing handle to widen the button so that it can accommodate the phrase “Please Push Me” on one line.

Alternately, you can drag the bottom sizing handle down and have the phrase displayed on two lines.

6. Run the program, and click on the button.

The color of the button darkens when the mouse hovers over it. In Section 2.3, we will write code that is executed when a button is clicked on.

7. End the program and select the button.

8. From the Properties window, edit the Text setting by inserting an ampersand (&) before the first letter *P*, and then press the Enter key.

Notice that the first letter *P* on the button is now underlined. See Fig. 2.14. Pressing Alt+*P* while the program is running causes the same event to occur as does clicking the button. Here, *P* is referred to as the **access key** for the button. (The access key is always the character following the ampersand.)

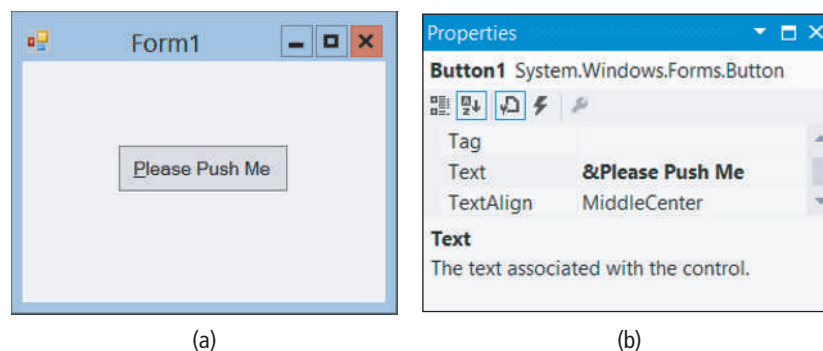



FIGURE 2.14 Designating P as an access key.


9. Click on *Close Solution* (or *Close Project*) in the FILE menu to close the program.

There is no need to save this program, so click on the *Discard* button.

■ A Label Walkthrough

1. Click on the *New Project* button to begin a new program.
Feel free to keep the default name, such as `WindowsApplication1`.
2. Double-click on the Label control ( **Label**) in the Toolbox to place a label on the form.
3. Drag the label to the center of the form.
4. Activate the Properties window, highlight the Text property, type “Enter Your Phone Number:”, and press the Enter key.
Such a label is placed next to a text box into which the user will type a phone number. Notice that the label widened to accommodate the text. This happened because the `AutoSize` property of the label is set to `True` by default.
5. Change the `AutoSize` property to `False` and press Enter.
Notice that the label now has eight sizing handles when selected.
6. Make the label narrower and longer until the words occupy two lines.
7. Activate the Properties window, and click on the down arrow to the right of the setting for the `TextAlign` property. Experiment by clicking on the various rectangles and observing their effects.
The combination of sizing and alignment permits you to design a label easily.
8. Run the program.
Nothing happens, even if you click on the label. Labels just sit there. The user cannot change what a label displays unless you write code to make the change.
9. End the program.
10. Click on *Close Solution* (or *Close Project*) in the FILE menu to close the program.
There is no need to save this program, so click on the *Discard* button.

■ A List Box Walkthrough

1. Click on the *New Project* button to begin a new program.
Feel free to keep the default name, such as `WindowsApplication1`.
2. Place a ListBox control ( **ListBox**) on the form.
3. Press F4 to activate the Properties window and notice that the list box does not have a Text property.
The word `ListBox1` that appears is actually the setting for the `Name` property.
4. Place a text box, a button, and a label on the form.
5. Click on the Object box just below the title bar of the Properties window.
The name of the form and the names of the four controls are displayed. If you click on one of the names, that object will become selected and its properties displayed in the Properties window.
6. Run the program.
Notice that the word `ListBox1` has disappeared, but the words `Button1` and `Label1` are still visible. The list box is completely blank. In subsequent sections, we will write code to place information into the list box.
7. End the program.
8. Click on *Close Solution* (or *Close Project*) in the FILE menu to close the program.
There is no need to save this program, so click on the *Discard* button.

■ The Name Property

The form and each control on it has a Name property. By default, the form is given the name Form1 and controls are given names such as TextBox1 and TextBox2. These names can (and should) be changed to descriptive ones that reflect the purpose of the form or control. Also, it is a good programming practice to have each name begin with a three-letter prefix that identifies the type of the object. See Table 2.1.

TABLE 2.1 Some three-letter prefixes.

Object	Prefix	Example
form	frm	frmPayroll
button	btn	btnComputeTotal
label	lbl	lblAddress
list box	lst	lstOutput
text box	txt	txtCity

The Solution Explorer window contains a file named Form1.vb that holds information about the form. Form1 is also the setting of the form's Name property in the Properties window. If you change the base name of the file Form1.vb, the setting of the Name property will automatically change to the new name. To make the change, right-click on Form1.vb in the Solution Explorer window, click on *Rename* in the context menu that appears, type in a new name (such as frmPayroll.vb), and press the Enter key. **Important:** Make sure that the new filename keeps the extension *vb*.

The name of a control placed on a form is changed from the control's Properties window. (The Name property is always the third property in the alphabetized list of properties.) Names of controls and forms must begin with a letter and can include numbers, letters, and underscore (_) characters, but cannot include punctuation marks or spaces.

The Name and Text properties of a button are both initially set to something like Button1. However, changing one of these properties does not affect the setting of the other property, and similarly for the Name and Text properties of forms, text boxes, and labels. The Text property of a form specifies the words appearing in the form's title bar.


■ Fonts

The default font for controls is Microsoft Sans Serif. Courier New is another commonly used font. Courier New is a fixed-width font; that is, each character has the same width. With such a font, the letter i occupies the same space as the letter m. Fixed-width fonts are used with tables when information is to be aligned in columns.

■ Auto Hide

The Auto Hide feature allows you to make more room on the screen for the Document window by hiding windows (such as the Toolbox, Solution Explorer, and Properties windows). Let's illustrate the feature with a walkthrough using the Toolbox window.

1. If the Toolbox window is not visible, click on *Toolbox* in the Menu bar's VIEW menu to see the window.

Auto Hide is active when the pushpin icon is horizontal (). When the Auto Hide feature is enabled, the Toolbox window will slide out of view when not needed.

2. If the pushpin icon is vertical (📌), then click on the icon to make it horizontal.

The Auto Hide feature is now enabled.

3. Move the mouse cursor somewhere outside the Toolbox window and click the left mouse button.

The window slides into a tab captioned Toolbox on the left side of the screen.

4. Hover the mouse cursor over the tab.

The window slides into view and is ready for use. After you click outside the window, it will return back into the tab.

5. Click on the pushpin icon to make it vertical.

The Auto Hide feature is now *disabled*.

6. Click the mouse cursor somewhere outside the Toolbox window.

The Toolbox window stays fixed. **Note:** We recommend keeping Auto Hide disabled for the Toolbox, Solution Explorer, and Properties windows unless you are creating a program with a very large form and need extra space.



VideoNote
Basic
Controls,
Sizing and
Aligning

■ Positioning and Aligning Controls

Visual Basic provides several tools for positioning and aligning controls on a form. **Proximity lines** are short line segments that help you place controls a comfortable distance from each other and from the sides of the form. **Snap lines** are horizontal and vertical line segments that help you align controls. The FORMAT menu is used to align controls, center controls horizontally and vertically in a form, and make a group of selected controls the same size.

A Positioning and Aligning Walkthrough

1. Begin a new program.
2. Place a button near the center of the form.
3. Drag the button toward the upper-right corner of the form until two short line segments appear. The line segments are called **proximity lines**. See Fig. 2.15(a). The button is now a comfortable distance from each of the two sides of the form.

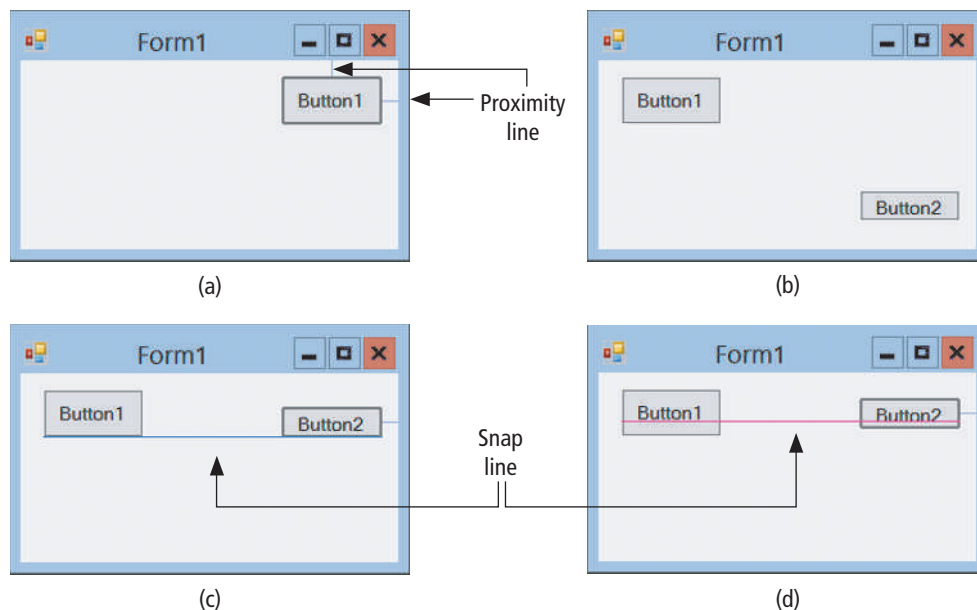


FIGURE 2.15 Positioning controls.

4. Place a second button below the first button and drag it upward until a proximity line appears between the two buttons.

The buttons are now a comfortable distance apart.

5. Resize and position the two buttons as shown in Fig. 2.15(b).
6. Drag Button2 upward until a blue line appears along the bottoms of the two buttons.
See Fig. 2.15(c). This blue line is called a **snap line**. The bottoms of the two buttons are now aligned.
7. Continue dragging Button2 upward until a purple snap line appears just underneath the words Button1 and Button2.

See Fig. 2.15(d). The middles of the two buttons are now aligned. If we were to continue dragging Button2 upward, a blue snap line would tell us when the tops were aligned. Steps 8 and 9 present another way to align the tops of the controls.

8. Click on Button1 and then hold down the Ctrl key and click on Button2.
After the mouse button is released, both buttons will be selected. **Note:** This process (called **selection of multiple controls**) can be repeated to select a group of any number of controls.
9. With the two buttons still selected, open the FORMAT menu in the Menu bar, hover over *Align*, and click on *Tops*.

The tops of the two buttons are now aligned. Precisely, Button1 (the first button selected) will stay fixed, and Button2 will move up so that its top is aligned with the top of Button1. The *Align* submenu also is used to align middles or corresponding sides of a group of selected controls. Some other useful submenus of the FORMAT menu are as follows:

Make Same Size: Equalize the width and/or height of the controls in a group of selected controls.

Center in Form: Center a selected control either horizontally or vertically in a form.

Vertical Spacing: Equalize the vertical spacing between a column of three or more selected controls.

Horizontal Spacing: Equalize the horizontal spacing between a row of three or more selected controls.

10. With the two buttons still selected, open the Properties window and set the ForeColor property to blue.

Notice that the ForeColor property has been altered for both buttons. Actually, any property that is common to every control in a group of selected multiple controls can be set simultaneously for the entire group of controls.

■ Multiple Controls

When a group of controls are selected with the Ctrl key, the first control selected (called the **master control** of the group) will have white sizing handles, while the other controls will have black sizing handles. All alignment and sizing statements initiated from the FORMAT menu will keep the master control fixed and will align (or size) the other controls with respect to the master control. You can designate a different control to be the master control by clicking on it.

After multiple controls have been selected, they can be dragged, deleted, and have properties set as a group. The arrow keys also can be used to move and size multiple controls as a group.

A group of multiple controls also can be selected by clicking the mouse outside the controls, dragging it across the controls, and releasing it. The *Select All* command from the EDIT menu (or the key combination Ctrl+A) causes all the controls on the form to be selected.



VideoNote
Using
Multiple
Controls

■ Setting Tab Order

Whenever the Tab key is pressed while a program is running, the focus moves from one control to another. The following walkthrough explains how to determine the order in which the focus moves and how that order can be changed.

1. Start a new program.
2. Place a button, a text box, and a list box on a form.
3. Run the program, and successively press the Tab key.

Notice that the controls receive the focus in the order they were placed on the form.

4. End the program.
5. Click on *Tab Order* in the VIEW menu.

The screen appears as in Fig. 2.16(a). The controls are numbered from 0 to 2 in the order they were created. Each of the numbers is referred to as a **tab index**.

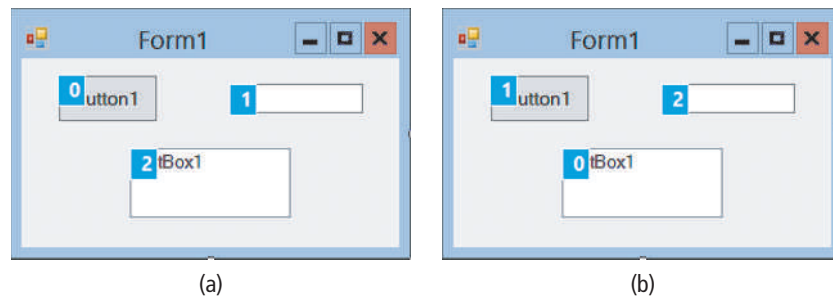


FIGURE 2.16 Tab order.

6. Click on the list box, then the button, and finally the text box.
7. Click again on *Tab Order* in the VIEW menu to set the new tab order.
8. Run the program again, and successively press the Tab key.

Notice that the controls receive the focus according to the new tab order.

9. End the program.
10. Add a label to the form, rerun the program, and successively press the Tab key.

Notice that the label does not receive the focus. Whether or not a control can receive the focus is determined by the setting of its `TabStop` property. By default, the setting of the `TabStop` property is `True` for buttons, text boxes, and list boxes, and `False` for labels. In this book we always use these default settings. **Note:** Even though labels do not receive the focus while tabbing, they are still assigned a tab index.

■ Comments

1. While you are working on a program, the program resides in memory. Removing a program from memory is referred to as **closing** the program. A program is automatically closed when you begin a new program. Also, it can be closed directly with the *Close Solution* (or *Close Project*) command from the FILE menu.
2. Three useful properties that have not been discussed are the following:
 - (a) `BackColor`: This property specifies the background color for the form or a control.
 - (b) `Visible`: Setting the `Visible` property to `False` causes an object to disappear when the program is run. The object can be made to reappear with code.

- (c) Enabled: Setting the Enabled property of a control to False restricts its use. It appears grayed and cannot receive the focus. Controls sometimes are disabled temporarily when they are not needed in the current state of the program.
- 3. Most properties can be set or altered with code as the program is running instead of being preset from the Properties window. For instance, a button can be made to disappear with a line such as `Button1.Visible = False`. The details are presented in Section 2.3.
- 4. If you inadvertently double-click on a form, a window containing text will appear. (The first line is `Public Class Form1`.) This is the Code Editor, which is discussed in the next section. To return to the Form Designer, click on the tab at the top of the Document window labeled “Form1.vb [Design].”
- 5. We have seen two ways to place a control onto a form. Another way is to just click on the control in the Toolbox and then click on the location in the form where you would like to place the control. Alternatively, you can just drag the control from the Toolbox to the location in the form.
- 6. Figure 2.9 on page 46 shows a small down-arrow button on the right side of the Text property setting box. When you click on that button, a rectangular box appears. The setting for the Text property can be typed into this box instead of into the Settings box. This method of specifying the setting is especially useful when you want the button to have a multiline caption.
- 7. We recommend setting the `StartPosition` property of the form to `CenterScreen`. With this setting the form will appear in the center of the screen when the program is run.
- 8. Refer to Fig. 2.8 on page 45. If you click on the button at the right side of the Properties window's Object box, a list showing all the controls on the form will drop down. You can then click on one of the controls to make it the selected control.

Practice Problems 2.2

1. What is the difference between the Text and the Name properties of a button?
2. The first two group names in the Toolbox are *All Windows Forms* and *Common Controls*. How many groups are there?

EXERCISES 2.2

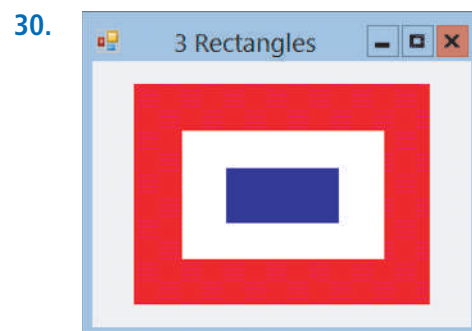
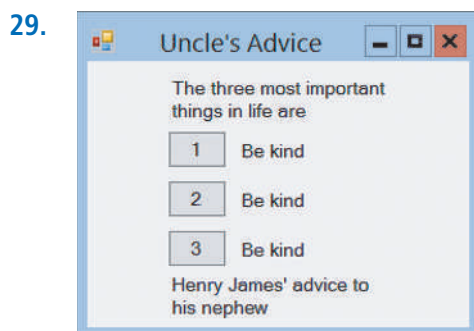
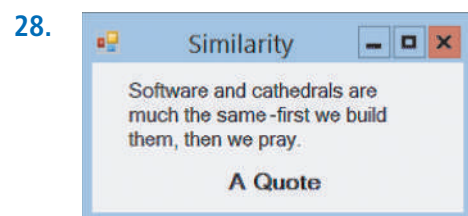
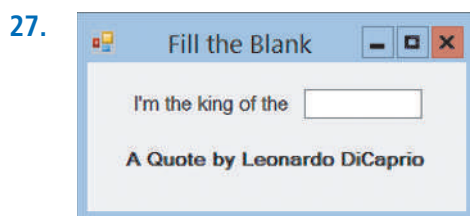
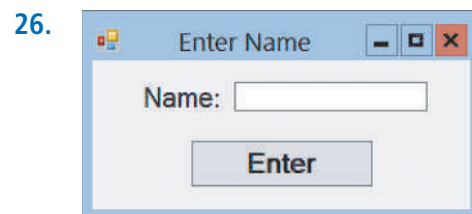
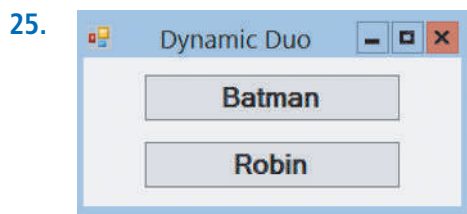
1. Create a form with a `ListBox` control, set the Name property to `lstFirst` and run the program. Do you notice anything different before and after running the program?
2. Name the different tools provided by Visual Basic for positioning and aligning controls on a form. Verify your answer by creating a form with two buttons.

In Exercises 3 through 24, carry out the task.

3. Create a form with background color yellow.
4. Alter the width of the textbox using the mouse.
5. Place the textbox center of the form using the mouse.
6. Create a form with a textbox. Activate the property window of the textbox using a single key from the keyboard.
7. Create a textbox `txtFirst` and display “Visual Basic” in Times New Roman with font size 12.
8. Create a form with a textbox and display “After all is said and done, more is said than done.” in the text box in bold, blue color.

9. Create a form with a title bar “Visual Basic” and a textbox with text “Visual Basic 2012” in Times New Roman.
10. Create a textbox txtFirst and display your first name in bold italic.
11. Create a textbox txtFirst and display your first name in the text box.
12. Create a textbox and find the default font, font style and font size.
13. Create a label with the text “Hello” in gold background.
14. Create a textbox txtFirst for output.
15. Equalize the vertical spacing between three textboxes.
16. Create a button btnFirst with the text “Please Push Me”.
17. Create a button and display the bold text “Please Push Me” in two lines. Each line should be center justified.
18. Create a button with the text “Save As”.
19. Create a button with the text “Send”, access key d.
20. Create two buttons and align the tops of the two buttons.
21. Create a Label lblFirst with the text “Enter Your Name:”.
22. Create a Label lblFirst with the two-line text “Enter Your First and Last Name”.
23. Create a Label lblFirst with bold, center aligned text “Register No.:”.
24. Create a List Box that will be disabled when the program is run.

In Exercises 25 through 30, create the form shown in the figure. (These exercises give you practice creating controls and assigning properties. The interfaces do not necessarily correspond to actual programs.)



31. Create a replica of your bank check on a form. Words common to all checks, such as “PAY TO THE ORDER OF”, should be contained in labels. Items specific to your checks, such as your name at the top left, should be contained in text boxes. Make the check on the screen resemble your personal check as much as possible. **Note:** Omit the account number.
32. Create a replica of your campus ID on a form. Words that are on all student IDs, such as the name of the college, should be contained in labels. Information specific to your ID, such as your name and student ID number, should be contained in text boxes.
33. Consider the form shown in Exercise 25. Assume the *Batman* button was added to the form before the *Robin* button. What is the tab index of the *Robin* button?
34. Consider the form shown in Exercise 26. Assume the first control added to the form was the label. What is the tab index of the label?

The following hands-on exercises develop additional techniques for manipulating and accessing controls placed on a form.

35. Place a label on a form and select the label. What is the effect of pressing the various arrow keys while holding down the Ctrl key?
36. Place a label on a form and select the label. What is the effect of pressing the various arrow keys while holding down the Shift key?
37. Repeat Exercise 36 for a Button.
38. Repeat Exercise 35 for a Button.
39. Create a label and a list box with font size 12 and font Times New Roman at the same time.
40. Place a Text Box txtexample in the center of a form and select it. Hold down the Ctrl key and press an arrow key. Repeat this process for each of the other arrow keys. Describe what happens.
41. Place a label and a list box on a form with the label to the left of and above the list box. Select the label. Hold down the Ctrl key and press the down-arrow key twice. With the Ctrl key still pressed, press the right-arrow key. Describe what happens.
42. Place two text boxes on a form with one text box to the right of and below the other text box. Select the lower text box, hold down the Ctrl key, and press the left-arrow key. With the Ctrl key still pressed, press the up-arrow key. Describe the effect of pressing the two arrow keys.
43. Explore FORMAT menu to determine the difference between the center and the middle of a control by placing a textbox and a button respectively on a form.
44. Place four different size textboxes vertically on a form. Use the FORMAT menu to make them the same size, center and to make the spacing between them uniform.
45. Place a label and a text box on a form as in Exercise 26, and then lower the label slightly and lower the text box until it is about one inch lower than the label. Use the mouse to slowly raise the text box to the top of the form. Three snap lines will appear along the way: a blue snap line, a purple snap line, and finally another blue snap line. What is the significance of each snap line?
46. Place a label on a form, select the label, and open its Properties window. Double-click on the name (not the Settings box) of the Visible property. Double-click again. What is the effect of double-clicking on a property whose possible settings are True and False?
47. Place a list box on a form, select the list box, and open its Properties window. Double-click on the name (not the Settings box) of the Enable property. Double-click repeatedly. Describe what is happening.

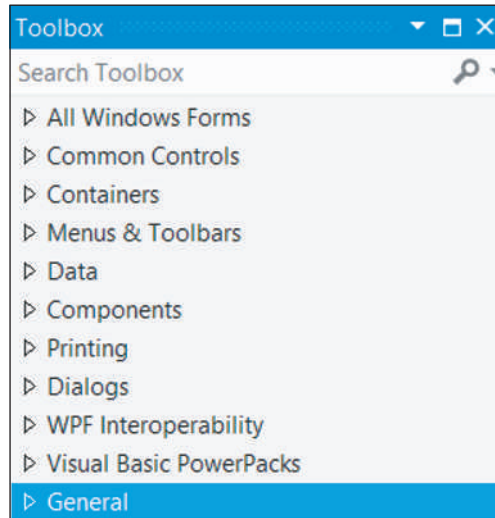


VideoNote

Moving a
Textbox
(Homework)

Solutions to Practice Problems 2.2

1. The text is the words appearing on the button, whereas the name is the designation used to refer to the button in code. Initially, they have the same value, such as Button1. However, each can be changed independently of the other.
2. The Toolbox in the Express Edition of Visual Basic contains 11 groups. Figure 2.17 shows the Toolbox after each group has been collapsed. **Note:** In the other editions of Visual Basic the Toolbox contains 12 groups.

**FIGURE 2.17** Toolbox group names.

VideoNote
Event
Procedures

2.3 Visual Basic Events

When a Visual Basic program runs, the form and its controls appear on the screen. Normally, nothing happens until the user takes an action, such as clicking a control or pressing a key. We call such an action an **event**. The programmer writes code that reacts to an event by performing certain tasks.

The three steps in creating a Visual Basic program are as follows:

1. Create the interface; that is, generate, position, and size the objects.
2. Set properties; that is, configure the appearance of the objects.
3. Write the code that executes when events occur.

Section 2.2 covered Steps 1 and 2; this section is devoted to Step 3. Code consists of statements that carry out tasks. Writing code in Visual Basic is assisted by an autocompletion system called **IntelliSense** that reduces the amount of memorization needed and helps prevent errors. In this section, we limit ourselves to statements that change properties of a control or the form while a program is running.

Properties of controls are changed in code with statements of the form

```
controlName.property = setting
```

where *controlName* is the name of the control, *property* is one of the properties of the control, and *setting* is a valid setting for that property. Such statements are called **assignment statements**. They assign values to properties. Here are three examples of assignment statements:

1. The statement

```
txtBox.Text = "Hello"
```

displays the word Hello in the text box.

2. The statement

```
btnButton.Visible = True
```

makes the button visible.

3. The statement

```
txtBox.ForeColor = Color.Red
```

sets the color of the characters in the text box named txtBox to red.

Most events are associated with controls. The event “click on btnButton” is different from the event “click on lstBox”. These two events are specified btnButton.Click and lstBox.Click. The statements to be executed when an event occurs are written in a block of code called an **event procedure** or **event handler**. The first line of an event procedure (called the **header**) has the form

```
Private Sub objectName_event(sender As System.Object,  
                             e As System.EventArgs) Handles objectName.event
```

Since we rarely make any use of the lengthy text inside the parentheses in this book, for the sake of readability we replace it with an ellipsis. However, it will automatically appear in our programs each time Visual Basic creates the header for an event procedure. The structure of an event procedure is

```
Private Sub objectName_event(...) Handles objectName.event  
    statements  
End Sub
```

where the three dots (that is, the ellipsis) represent

```
sender As System.Object, e As System.EventArgs
```

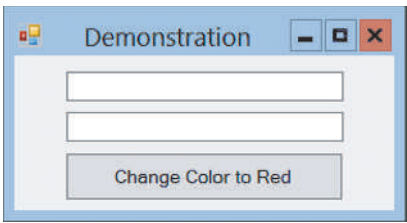
Words such as “Private,” “As,” “Sub,” “Handles,” and “End” have special meanings in Visual Basic and are referred to as **keywords** or **reserved words**. The Code Editor automatically capitalizes the first letter of a keyword and displays the word in blue. The word “Sub” in the first line signals the beginning of the procedure, and the first line identifies the object and the event occurring to that object. The last line signals the termination of the event procedure. The statements to be executed appear between these two lines. These statements are referred to as the **body** of the event procedure. (**Note:** The word “Private” indicates that the event procedure cannot be invoked by another form. This will not concern us until much later in the book. The expression following “Handles” identifies the object and the event happening to that object. The expression “objectName_event” is the default name of the procedure and can be changed if desired. In this book, we always use the default name. The word “Sub” is an abbreviation of *Subroutine*.) For instance, the event procedure

```
Private Sub btnButton_Click(...) Handles btnButton.Click  
    txtBox.ForeColor = Color.Red  
End Sub
```

changes the color of the words in the text box to red when the button is clicked. The clicking of the button is said to **raise** the event, and the event procedure is said to **handle** the event.

■ An Event Procedure Walkthrough

The form in Fig. 2.18, on the next page, which contains two text boxes and a button, will be used to demonstrate what event procedures are and how they are created. Three event procedures will be used to alter the appearance of a phrase appearing in a text box. The event procedures are named txtFirst_TextChanged, btnRed_Click, and txtFirst_Leave.



OBJECT	PROPERTY	SETTING
frmDemo	Text	Demonstration
txtFirst		
txtSecond		
btnRed	Text	Change Color to Red

FIGURE 2.18 The interface for the event procedure walkthrough.

1. Create the interface in Fig. 2.18 in the Form Designer. The Name properties of the form, text boxes, and button should be set as shown in the Object column. The Text property of the form should be set to Demonstration, and the Text property of the button should be set to Change Color to Red. No properties need be set for the text boxes.
2. Click the right mouse button anywhere on the Form Designer, and click on *View Code*. The Form Designer IDE is replaced by the **Code Editor** (also known as the *Code view* or the *Code window*). See Fig. 2.19.

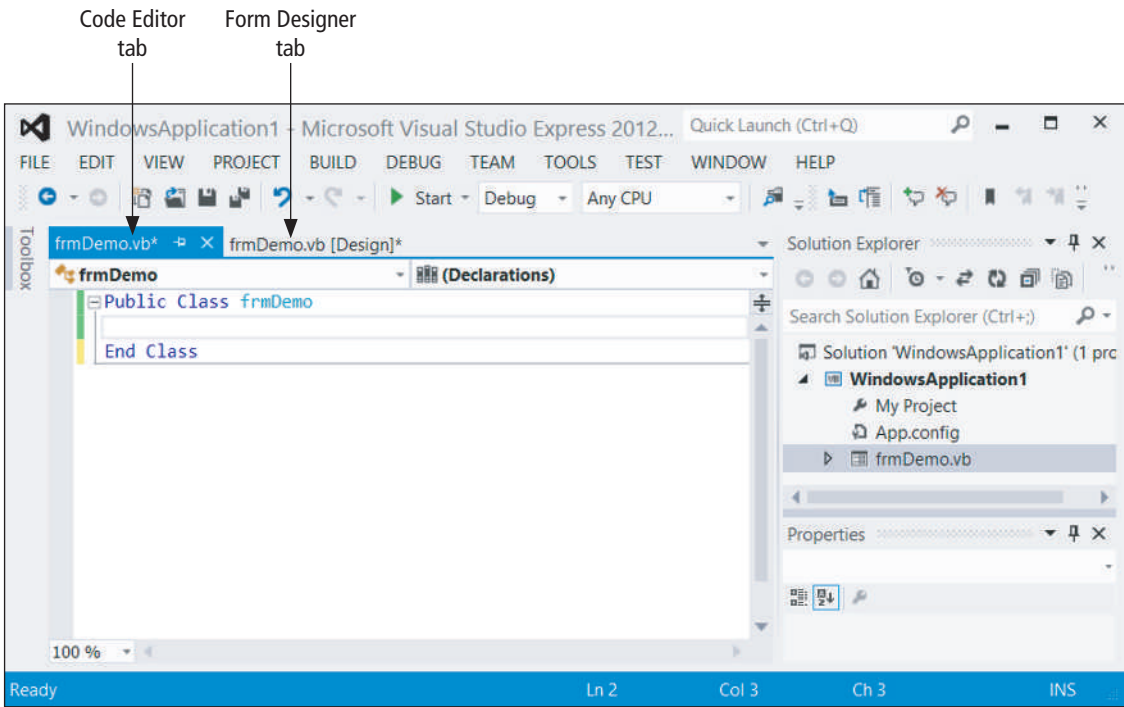


FIGURE 2.19 The Visual Basic IDE in Code Editor mode.

The tab labeled `frmDemo.vb` corresponds to the Code Editor. Click on the tab labeled `frmDemo.vb [Design]` when you want to return to the Form Designer. We will place our program code between the two lines shown.

Figure 2.19 shows that the Code Editor IDE has a Toolbox, Solution Explorer, and Properties window that support Auto Hide. The Solution Explorer window for the Code Editor functions exactly like the one for the Form Designer. The Code Editor's Toolbox has just one group, General, that is used to store code fragments that can be copied into a program when needed. The Code Editor's Properties window will not be used in this textbook.

3. Click on the tab labeled “`frmDemo.vb [Design]`” to return to the Form Designer. (You also can invoke the Form Designer by clicking on *Designer* in the *VIEW* menu, or by right-clicking the Code Editor and clicking on *View Designer*.)

4. Double-click on the button. The Code Editor reappears, but now the following two lines of code have been added to it and the cursor is located on the blank line between them.

```
Private Sub btnRed_Click(...) Handles btnRed.Click
End Sub
```

The first line is the header for an event procedure named `btnRed_Click`. This procedure is invoked by the event `btnRed.Click`. That is, whenever the button is clicked, the code between the two lines just shown will be executed.

5. Type the line

```
txtFirst.ForeColor = Color.Red
```

at the cursor location.

This statement begins with the name of a control, `txtFirst`. Each time you type a letter of the name, IntelliSense drops down a list containing possible completions for the name.¹ As you continue typing, the list is shortened to match the letters that you have typed. Figure 2.20 shows the list after the letters `tx` have been typed. At this point, you have the following three options on how to continue:

- i. Double-click on `txtFirst` in the list.
- ii. Keep the cursor on `txtFirst`, and then press the Tab key or the Enter key.
- iii. Directly type in the remaining six letters of `txtFirst`.

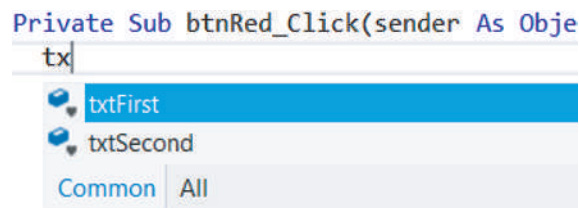


FIGURE 2.20 Drop-down list produced by IntelliSense.

After you type the dot (.) following `txtFirst`, IntelliSense drops down a list containing properties of text boxes. See Fig. 2.21(a) on the next page. Each property is preceded by a properties icon (🔧). [The list also contains items called *methods*, which we will discuss later. Methods are preceded by a method icon (⚙️).] At this point, you can scroll up the list and double-click on `ForeColor` to automatically enter that property. See Fig. 2.21(b). Or, you can keep typing. After you have typed “For”, the list shortens to the single word `ForeColor`. At that point, you can press the Tab key or the Enter key, or keep typing to obtain the word `ForeColor`.

After you type in the equal sign, IntelliSense drops down the list of colors shown in Fig. 2.22. You have the option of scrolling to `Color.Red` and double-clicking on it, or typing `Color.Red` into the statement.

6. Return to the Form Designer and double-click on the first text box. The Code Editor reappears, and the first and last lines of the event procedure `txtFirst_TextChanged` appear in it. This procedure is raised by the event `txtFirst.TextChanged`—that is, whenever there is a change in the text displayed in the text box `txtFirst`. Type the line that sets the `ForeColor` property of `txtFirst` to blue. The event procedure will now appear as follows:

```
Private Sub txtFirst_TextChanged(...) Handles txtFirst.TextChanged
    txtFirst.ForeColor = Color.Blue
End Sub
```

¹This feature of IntelliSense is referred to as **Complete Word**.

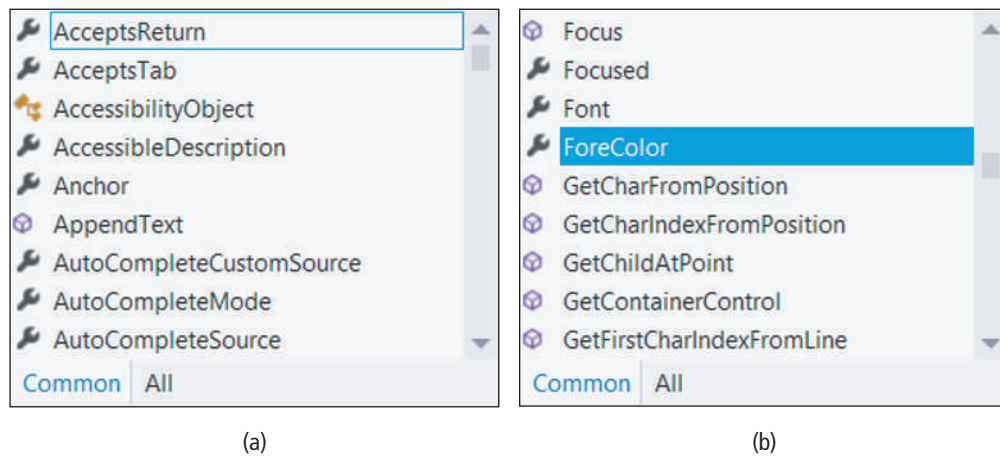


FIGURE 2.21 Drop-down list produced by IntelliSense.

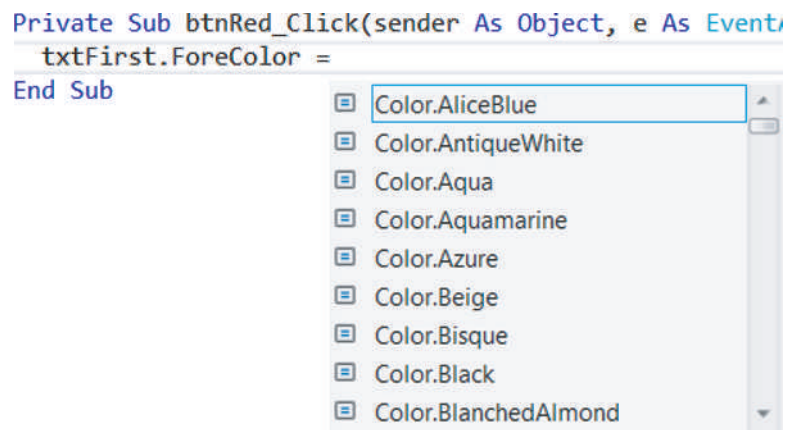


FIGURE 2.22 Drop-down list of colors produced by IntelliSense.

7. Return to the Form Designer and select txtFirst.
8. Click on the *Events* button (⚡) in the toolbar near the top of the Properties window. The 63 events associated with text boxes are displayed, and the Description pane at the bottom of the window describes the currently selected event. (Don't be alarmed by the large number of events. Only a few events are used in this book.) Scroll to the *Leave* event. See Fig. 2.23.

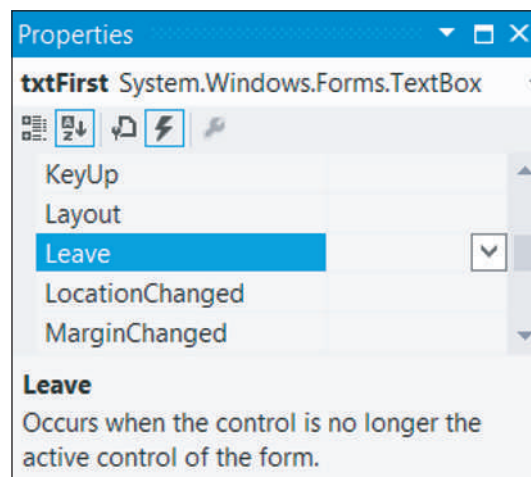


FIGURE 2.23 Events displayed in the Properties window.

9. Double-click on the Leave event. (The event txtFirst.Leave is raised when the focus is moved away from the text box.) The header and the last line of the event procedure txtFirst_Leave will be displayed. In this procedure, type the line that sets the ForeColor property of txtFirst to Black. The Code Editor will now look as follows:

```
Public Class frmDemo
    Private Sub btnRed_Click(...) Handles btnRed.Click
        txtFirst.ForeColor = Color.Red
    End Sub

    Private Sub txtFirst_Leave(...) Handles txtFirst.Leave
        txtFirst.ForeColor = Color.Black
    End Sub

    Private Sub txtFirst_TextChanged(...) Handles txtFirst.TextChanged
        txtFirst.ForeColor = Color.Blue
    End Sub
End Class
```

10. Hover the cursor over the word “ForeColor”. Visual Basic now displays information about the foreground color property. This illustrates another help feature of Visual Basic.
11. Run the program by pressing F5.
12. Type something into the first text box. In Fig. 2.24, the blue word “Hello” has been typed. (Recall that a text box has the focus whenever it is ready to accept typing—that is, whenever it contains a blinking cursor.)

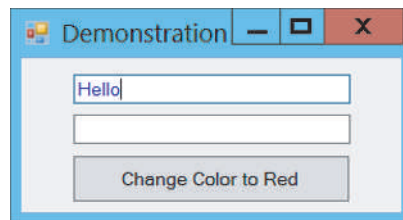



FIGURE 2.24 Text box containing input.

13. Click on the second text box. The contents of the first text box will become black. When the second text box was clicked, the first text box lost the focus; that is, the event Leave happened to txtFirst. Thus, the event procedure txtFirst_Leave was invoked, and the code inside the procedure was executed.
14. Click on the button. This invokes the event procedure btnRed_Click, which changes the color of the words in txtFirst to red.
15. Click on the first text box, and type the word “Friend” after the word “Hello”. As soon as typing begins, the text in the text box is changed and the TextChanged event is raised. This event causes the color of the contents of the text box to become blue.
16. You can repeat Steps 12 through 15 as many times as you like. When you are finished, end the program by clicking on the *Stop Debugging* button on the Toolbar, clicking on the form’s *Close* button, or pressing Alt+F4.

Note: After viewing events in the Properties window, click on the *Properties* button () to the left of the *Events* button to return to displaying properties in the Properties window.

■ Properties and Event Procedures of the Form

You can assign properties to the form itself in code. However, a statement such as

```
frmDemo.Text = "Demonstration"
```


will not work. The form is referred to by the keyword *Me*. Therefore, the proper statement is

```
Me.Text = "Demonstration"
```

To display a list of all the events associated with *frmDemo*, select the form in the Form Designer and then click on the *Events* button in the Properties window's toolbar.

■ The Header of an Event Procedure

As mentioned earlier, in the header for an event procedure such as

```
Private Sub btnOne_Click(...) Handles btnOne.Click
```

btnOne_Click is the name of the event procedure, and *btnOne.Click* identifies the event that invokes the procedure. The name can be changed at will. For instance, the header can be changed to

```
Private Sub ButtonPushed(...) Handles btnOne.Click
```

Also, an event procedure can handle more than one event. For instance, if the previous line is changed to

```
Private Sub ButtonPushed(...) Handles btnOne.Click, btnTwo.Click
```

the event procedure will be invoked if either *btnOne* or *btnTwo* is clicked.

We have been using ellipses (...) as place holders for the phrase

```
sender As System.Object, e As System.EventArgs
```

In Chapter 5, we will gain a better understanding of this type of phrase. Essentially, the word “sender” carries a reference to the object that raised the event, and the letter “e” carries some additional information that the sending object wants to communicate. We will make use of “sender” and/or “e” in Section 9.4 only when sending output to the printer. You can delete the entire phrase from any other type of program in this book and just leave the blank set of parentheses.

■ Opening a Program

Beginning with the next chapter, each example contains a program. These programs can be downloaded from the Pearson website for this book. See the discussion on page 17 of the Preface for details. The process of loading a program stored on a disk into the Visual Basic environment is referred to as **opening** the program. Let's open the downloaded program 7-2-3 from Chapter 7. That program allows you to enter a first name, and then displays U.S. presidents having that first name.

1. From Visual Basic, click on *Open Project* in the FILE menu. (An Open Project dialog box will appear.)
2. Navigate to the contents of the Ch07 subfolder downloaded from the website.
3. Double-click on 7-2-3.
4. Double-click on 7-2-3.sln.
5. If the Solution Explorer window is not visible, click on *Solution Explorer* in the VIEW menu.
6. If the file *frmPresident.vb* is not visible in the Solution Explorer, click on the symbol to the left of 7-2-3 in the Solution Explorer in order to display the file.

7. Double-click on `frmPresident.vb`. The Form Designer for the program will be revealed and the Solution Explorer window will appear as in Fig. 2.25. (You can click on the *View Code* button to reveal the Code Editor. The *Show All Files* and *Refresh* buttons, which allow you to view all the files in a program's folder and to update certain files, will be used extensively beginning with Chapter 7.)

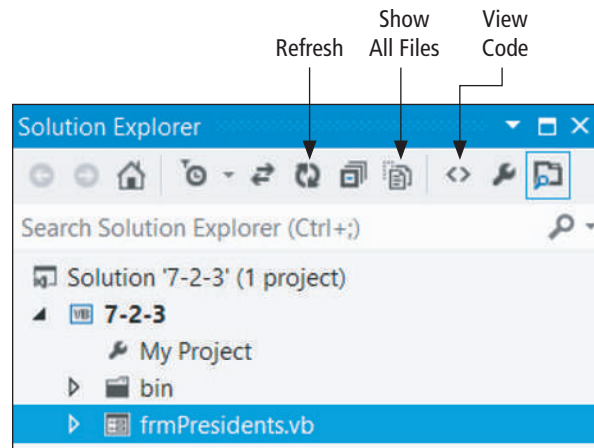


FIGURE 2.25 Solution Explorer window.

8. Press F5 to run the program.
9. Type in a name (such as James or William), and press the *Display Presidents* button. (See Fig. 2.26.) You can repeat this process as many times as desired.

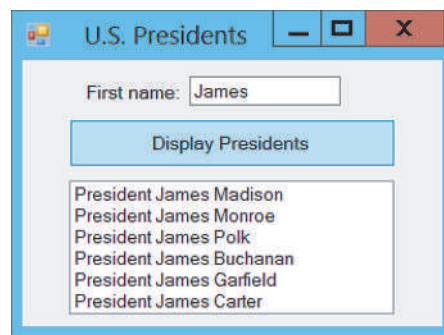


FIGURE 2.26 Output for program 7-2-3.

10. End the program.

The program just executed uses a text file named `USPres.txt`. To view the text file, open the folder `bin`, open the subfolder `Debug`, and click on `USPres.txt`. (If the `bin` folder is not visible, click on the *Show All Files* button. If `USPres.txt` is not listed in the `Debug` subfolder, click the *Refresh* button and reopen the folders. After reading Chapter 7, you will understand why text files are placed in the `Debug` subfolder of the `bin` folder.) The first line of the file gives the name of the first president; the second line gives the name of the second president, and so on. To close the text file, click on the *Close* button (X) on the `USPres.txt` tab.

Comments

1. The Visual Basic editor automatically indents the statements inside procedures. In this book, we indent by two spaces. To instruct your editor to indent by two spaces, click on *Options* in the **TOOLS** menu to display an Options dialog box, expand the “Text Editor” item in the

left pane, expand the “Basic” item, click on “Tabs”, enter 2 into the “Indent size:” box, and click on the OK button.

2. The event `controlName.Leave` is raised when the specified control loses the focus. Its counterpart is the event `controlName.Enter` which is raised when the specified control gets the focus. A related statement is

```
controlName.Focus()
```

which moves the focus to the specified control.

3. We have ended our programs by clicking the *Stop Debugging* button or pressing Alt+F4. A more elegant technique is to create a button, call it `btnQuit`, with caption *Quit* and the following event procedure:

```
Private Sub btnQuit_Click(...) Handles btnQuit.Click
    Me.Close()
End Sub
```

4. For statements of the form

```
object.Text = setting
```

the expression for *setting* must be surrounded by quotation marks. (For instance, the statement might be `lblName.Text = “Name:”`.) For properties where the proper setting is one of the words *True* or *False*, these words should *not* be surrounded by quotation marks.

5. Names of existing event procedures associated with an object are not automatically changed when you rename the object. You must change them yourself. However, the event that invokes the procedure (and all other references to the control) will change automatically. For example, suppose an event procedure is

```
Private Sub btnOne_Click(...) Handles btnOne.Click
    btnOne.Text = "Press Me"
End Sub
```

and, in the Form Designer, you change the name of `btnOne` to `btnTwo`. Then, when you return to the Code Editor, the procedure will be

```
Private Sub btnOne_Click(...) Handles btnTwo.Click
    btnTwo.Text = "Press Me"
End Sub
```

6. The Code Editor has many features of a word processor. For instance, the operations cut, copy, paste, and find can be carried out from the *EDIT* menu.
7. The Code Editor can detect certain types of errors. For instance, if you type

```
txtFirst.Text = hello
```

and then move away from the line, the automatic syntax checker will underline the word “hello” with a blue squiggle to indicate that something is wrong. When the mouse cursor is hovered over the underlined expression, the editor will display a message explaining what is wrong. If you try to run the program without correcting the error, the dialog box in Figure 2.27 will appear.

8. Each control has a favored event, called the **default event**, whose event procedure template can be generated from the Form Designer by double-clicking on the control. Table 2.2 shows some controls and their default events. The most common event appearing in this book is the *Click* event for a button. The *TextChanged* event for a text box was used in this section.

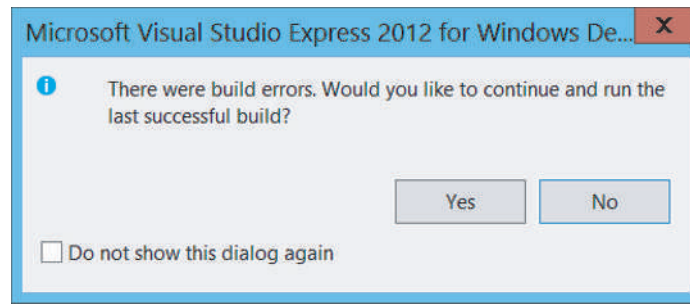


FIGURE 2.27 Error dialog box.

TABLE 2.2 Some default events.

Control	Default Event
form	Load
button	Click
label	Click
list box	SelectedIndexChanged
text box	TextChanged

The SelectedIndexChanged event for a list box is introduced in Section 4.4, and the Load event for a form is introduced in Section 7.1. The Click event for a label is never used in this book.

9. Font properties, such as the name, style, and size, are usually specified at design time. The setting of the properties can be displayed in code with statements such as

```
lstBox.Items.Add(txtBox.Font.Name)
lstBox.Items.Add(txtBox.Font.Bold)
lstBox.Items.Add(txtBox.Font.Size)
```

However, a font's name, style, and size properties cannot be altered in code with statements of the form

```
txtBox.Font.Name = "Courier New"
txtBox.Font.Bold = True
txtBox.Font.Size = 16
```

10. When you make changes to a program, asterisks appear as superscripts on the tabs labeled "frmName.vb [design]" and "frmName.vb" to indicate that some part of the program has not been saved. The asterisks disappear when the program is saved or run.

Note: If the program has been saved to disk, all files for the program will be automatically updated on the disk whenever the program is saved or run.

11. You can easily change the size of the font used in the current program's Code Editor. Just hold down the Ctrl key and move the mouse's scroll wheel.
12. Notes on IntelliSense:

- (a) Whenever an item in an IntelliSense drop-down list is selected, a tooltip describing the item appears to the right of the item.
- (b) From the situation in Fig. 2.20, on page 61, we can display txtFirst by double-clicking on the highlighted item, pressing the Tab key, or pressing the Enter key. Another option

is to press the period key. In this case, both the name `txtFirst` and the dot following it will be displayed. **Note:** The period key option works only if the selected item is always followed by a dot in code.

- (c) IntelliSense drop-down lists have tabs labeled *Common* and *All*. When the *All* tab is selected, every possible continuation choice appears in the list. When the *Common* tab is selected, only the most frequently used continuation choices appear.
- (d) Occasionally, the IntelliSense drop-down list will cover some of your program. If you hold down the `Ctrl` key, the drop-down list will become transparent and allow you to see the covered-up code.

Practice Problems 2.3

1. Describe the event that invokes the following event procedure.

```
Private Sub btnCompute_Click(...) Handles txtBox.Leave
    txtBox.Text = "Hello world"
End Sub
```

2. Give a statement that will prevent the user from typing into `txtBox`.

EXERCISES 2.3

In Exercises 1 through 6, describe the contents of the text box after the button is clicked.

1.

```
Private Sub btnOutput_Click(...) Handles btnOutput.Click
    txtBox.Visible = False
End Sub
```
2.

```
Private Sub btnOutput_Click(...) Handles btnOutput.Click
    txtBox.Text = "Hello"
    txtBox.Visible = False
End Sub
```
3.

```
Private Sub btnOutput_Click(...) Handles btnOutput.Click
    txtBox.Text = "Goodbye"
    txtBox.BackColor = Color.Orange
End Sub
```
4.

```
Private Sub btnOutput_Click(...) Handles btnOutput.Click
    txtBox.Text = "Hello"
    txtBox.ForeColor = Color.Yellow
End Sub
```
5.

```
Private Sub btnOutput_Click(...) Handles btnOutput.Click
    txtBox.Text = "Hello"
    txtBox.Enable = False
End Sub
```
6.

```
Private Sub btnOutput_Click(...) Handles btnOutput.Click
    txtBox.BackColor = Color.Yellow
    txtBox.Visible = True
End Sub
```