An Introduction to Programming Using Visual Basic®

An Introduction to Programming Using Visual Basic®

Eleventh Edition

David I. Schneider

University of Maryland

Pearson

Senior Vice President Courseware Portfolio Management: Marcia J. Horton
Director, Portfolio Management: Engineering, Computer Science & Global Editions: Julian Partridge
Executive Portfolio Manager: Tracy Johnson
Portfolio Management Assistant: Meghan Jacoby
Managing Content Producer: Scott Disanno
Content Producer: Carole Snyder
Rights and Permissions Manager: Ben Ferrini
Manufacturing Buyer, Higher Ed, Lake Side Communications, Inc. (LSC): Maura Zaldivar-Garcia and Deidra Smith
Inventory Manager: Bruce Boundy
Product Marketing Manager: Yvonne Vannatta
Field Marketing Manager: Demetrius Hall
Marketing Assistant: Jon Bryant
Cover Designer: Black Horse Design, Inc.
Cover Photo: Liudmila Habrus/123RF
Printer/Binder: LSC Communications, Inc.
Full-Service Project Management: Mahalakshmi Narayanan, Integra Software Services Pvt. Ltd.

Library of Congress Cataloging-in-Publication Data

Names: Schneider, David I., author.
Title: An introduction to programming using Visual Basic / David I. Schneider, University of Maryland.
Description: Eleventh edition. | Pearson, [2019] | Includes bibliographical references and index.
Identifiers: LCCN 2018051617| ISBN 9780135416037 | ISBN 0135416035
Subjects: LCSH: BASIC (Computer program language) | Visual Basic.
Classification: LCC QA76.73.B3 S333633 2019 | DDC 005.26/8–dc23
LC record available at https://lccn.loc.gov/2018051617

Pearson

ISBN – 10: 0-13-541603-5
ISBN – 13: 978-0-13-541603-7

Attention Students

Installing Visual Studio

To complete the tutorials and programming problems in this book, you need to install Visual Studio 2017 on your computer.

We recommend that you download Visual Studio Community 2017 from the following Web site, and install it on your system:

www.visualstudio.com

Visual Studio Community 2017 is a free, full-featured development environment, and is a perfect companion for this textbook.

NOTE: If you are working in your school's computer lab, there is a good chance that Microsoft Visual Studio has already been installed. If this is the case, your instructor will show you how to start Visual Studio.

Installing the Student Sample Program Files

The Student Sample Program files that accompany this book are available for download from the book's companion Web site at:

www.pearson.com/cs-resources

These files are required for many of the book's tutorials. Simply download the Student Sample Program files to a location on your hard drive where you can easily access them.

VideoNote

Guide to VideoNotes

www.pearson.com/cs-resources

Guide to Application Topics

Business and Economics

General Interest

Mathematics

Sports and Games

Contents

PREFACE

Visual Basic has been a widely used programming language since its introduction in 1991. Its latest incarnation, Visual Basic 2017, 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 2017 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 Eleventh Edition

1. ***Visual Basic Upgraded*** The version of Visual Basic has been upgraded from Visual Basic 2015 to Visual Basic 2017.
2. ***Updated Screen Captures*** One hundred and forty-five screen captures have been replaced with up-to-date versions.
3. ***New IntelliSense Features*** The new-to-VB2017 IntelliSense features are used.
4. ***Run to Click*** The new-to-VB2017 debugging feature "Run to Click" is explained and used in a walkthrough.
5. ***Revised Databases*** The databases used in 35 exercises have been updated or expanded.
6. ***Inside Back Cover*** The instructions on how to carry out 13 important Visual Basic operations are presented on the inside back cover.

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.

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.

Captions. Every example and applied exercise is labeled with a caption identifying its type of application.

Screen Captures. The output for most applied exercises and programming projects are shown in screen captures. This feature helps clarify the intent of the exercises.

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 Web site at http://www.pearsonhighered.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. Thirty VideoNotes are available at www.pearson.com/cs-resources. 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 MyLab Programming helps students fully grasp the logic, semantics, and syntax of programming. Through practice exercises and immediate, personalized feedback, MyLab Programming improves the programming competence of beginner students who struggle with the basic concepts and paradigms of popular high-level programming languages.

A self-study and homework tool, the MyLab Programming course for Visual Basic consists of roughly two hundred short practice exercises covering introductory topics such as variables, calculations, decision statements, loops, procedures, arrays, and more. 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 MyLab Programming in your course, ask for your Pearson representative or visit www.pearson.com/mylab/programming.

Instructor Resources

The following protected instructor resource materials are available on the Publisher's Web site at www.pearson.com/cs-resources. 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 companion Web site is located at www.pearson.com/cs-resources.

The following content is available through the companion 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 of this text and I am most grateful for their contributions. The current edition benefited greatly from the valuable comments of the following reviewers:

Laura Dinsmoor, Oakland University
Katie Gray, The University of Texas at Austin
Jason Iorio, University of Hartford
Uzo Osuno, Nassau Community College

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 Content Producer Carole Snyder and Managing Content Producer Scott Disanno. The competence and graciousness of Mahalakshmi Narayanan at Integra Software Services Pvt. Ltd. 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
dis@math.umd.edu

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 2017
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 Procedures[1]

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 Files[2]

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

[1] Passing 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.

[2] Sections 8.1 and 8.2 are independent of each other.

An Introduction to Programming Using Visual Basic®

1 An Introduction to Computers and Problem Solving

1.1 An Introduction to Computing and Visual Basic

An Introduction to Programming Using Visual Basic is about problem solving using computers. The programming language used is Visual Basic 2017 (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 *high-level language*. It consists of instructions to which people can relate, such as Click, If, and Do. Some other well-known high-level languages are Java, C++, and Python.

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

Answer: Tasks are broken down into a set of instructions, called a *program*, that can be expressed in a computer language. 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 is a GUI?*

Answer: What the user views on the monitor and interacts with while a program is running is called the *user interface*. 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: *What are the meanings of the terms "programmer" and "user"?*

Answer: A *programmer* (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: *What is the meaning of the term "code"?*

Answer: The Visual Basic instructions that the programmer writes are called *code*. The process of writing a program is often called *coding*.

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

Answer: Most programs do three things: take in data, manipulate data, and produce results. These operations are referred to as *input*, *processing*, and *output*. The input data might be held in the program, reside on a disk, 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 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 the sales tax rate. The 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, numbering 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?*

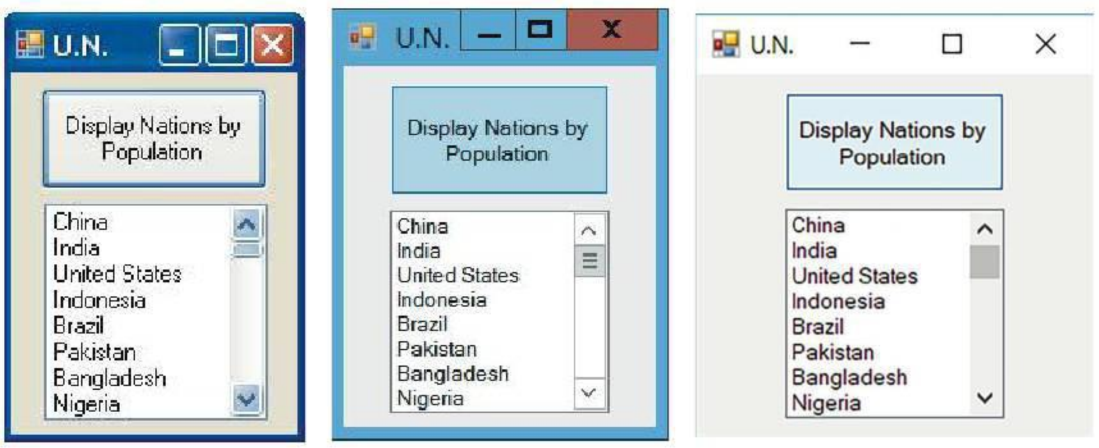
Answer: Problems are solved by carefully reading them to determine what data are given and what outputs are requested. Then a step-by-step procedure is devised to process the given data and produce the requested output.

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

Answer: You should be familiar with how folders (also called *directories*) and files are managed by Windows. Files reside on storage devices such as hard disks, USB flash drives, CDs, and DVDs. Traditionally, the primary storage devices for personal computers were hard disks and floppy disks. Therefore, the word *disk* is frequently used to refer to any storage device.

Question: *Will it matter whether Windows 7, Windows 8, or Windows 10 are used as the underlying operating system?*

Answer: Visual Basic runs fine with all three of these versions of Windows. (When using Windows 7, Service Pack 1 must also be installed.) 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 depends on the Windows product edition, the hardware on your system, and your own personal preferences. In this book, all screen captures have been done with the Windows 10 operating system.



(a) Windows 7 (b) Windows 8 (c) Windows 10

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.

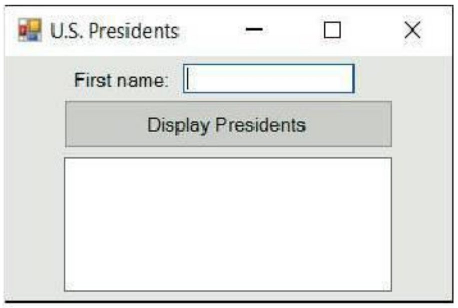
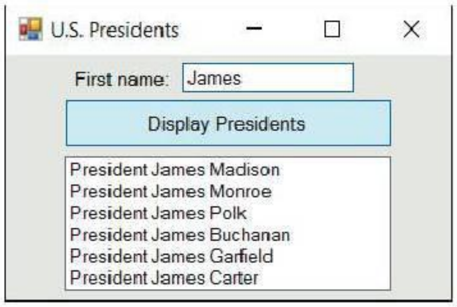


FIGURE 1.2 Window when program is first run.



```
President James Madison
President James Monroe
President James Polk
President James Buchanan
President James Garfield
President James Carter
```

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".

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), the code, and a file containing the data.

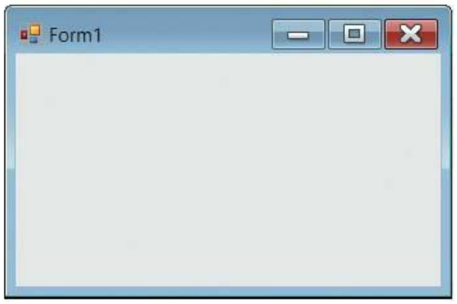


FIGURE 1.4 A blank Visual Basic form.

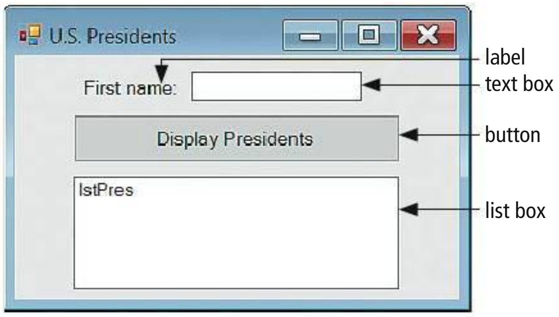


FIGURE 1.5 Controls added to the form.

Question: *What conventions are used to show keystrokes?*

Answer: The combination *key*1+*key*2 means "hold down key1 and then press key2". The combination Ctrl+C places selected material into the Clipboard. The combination *key1/key2* means "press and 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: *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: Possibly. By default, Windows 7 and Windows 8 show only the base names of files. You should configure Windows to display the filename extensions for all known file types. (The details are presented in Appendix B in the "Configuring the Windows Environment" section.) Windows 10 shows the full file name by default.

Question: *Will it matter what version of Visual Studio has been installed?*

Answer: Program outputs will look the same with all versions of Visual Studio. However, there will be slight differences in certain dialog boxes (such as the New Project dialog box) generated by Visual Basic that assist with the creation of programs.

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

Answer: Yes. Four adjustments are discussed in the textbook. In Section 2.2, a setting is specified that guarantees flexibility when 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 and using variables. In Section 7.2, we require that the installation of Visual Basic be modified to enable LINQ, a powerful tool that is used extensively in programs beginning in Chapter 7.

Question: *Where will new programs be saved?*

Answer: Before writing your first program, you should use File Explorer (with Windows 8 or 10) or Windows Explorer (with 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). The information needed to obtain the answer is the distance and time the car has traveled (the input). The formula

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

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

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

A graphical representation of this problem-solving process is shown in Fig. 1.6.

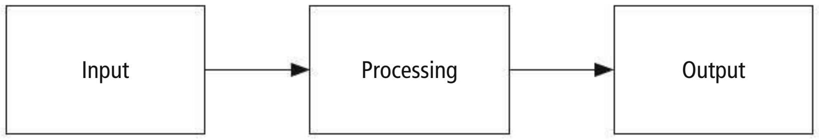


FIGURE 1.6 **The problem-solving process.**

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 perform the task. 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 hierarchy 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. If necessary, round the quotient up to a 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 Fig. 1.7.

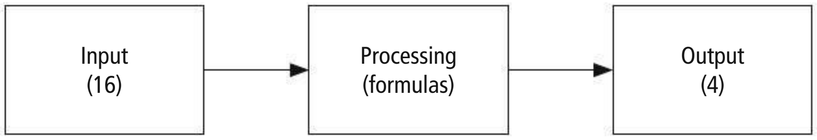


FIGURE 1.7 The problem-solving process for the stamp problem.

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

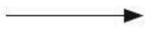
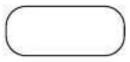
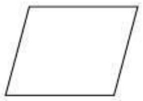
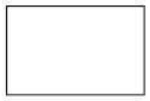
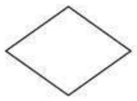
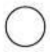
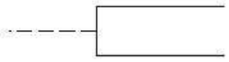
Flowcharts: Graphically depict the logical steps to carry out a task and show how the steps relate to each other.

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. The data to be input or output is described in the parallelogram.
	*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.

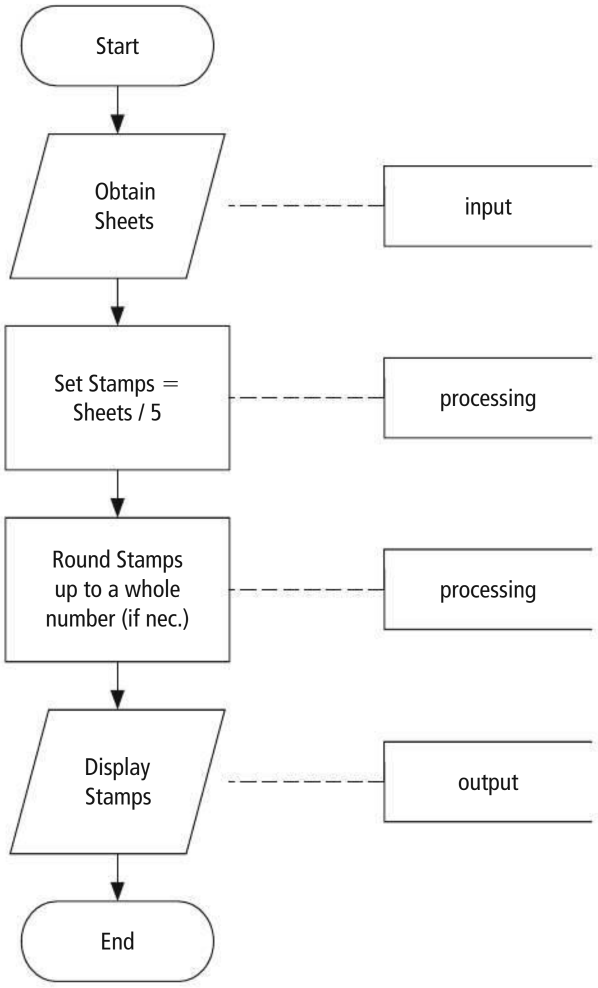


FIGURE 1.8 **Flowchart for the postage-stamp problem.**

The table of the flowchart symbols shown on the previous page has been adopted by the American National Standards Institute (ANSI). Figure 1.8 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, thereby making 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.
Obtain the number of sheets (Sheets)                          (input)
Set the number of stamps to Sheets / 5                        (processing)
Round the number of stamps up to a whole number (if nec.)     (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 most 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 of the program does and they show how the parts relate to each other. The details on how the parts work, however, are omitted. The chart is read from top to bottom and from left to right. Each part may be subdivided into a succession of subparts that branch out under it. Typically, after the activities in the succession of subparts are carried out, the part to the right of the original part is considered. A quick glance at the hierarchy chart reveals each task performed in the program and where it is performed. Figure 1.9 shows a hierarchy chart for the postage-stamp problem.

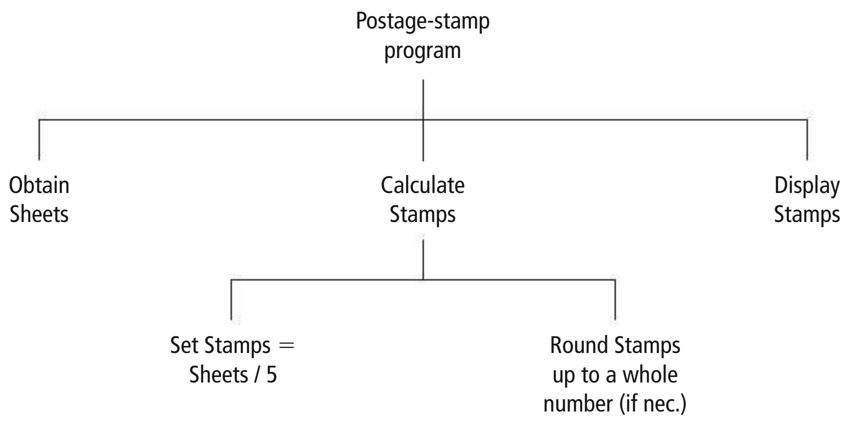


FIGURE 1.9 **Hierarchy chart for the postage-stamp problem.**

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 part 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

```
If condition is true Then
    Process step(s) 1
Else
    Process step(s) 2
End If
```

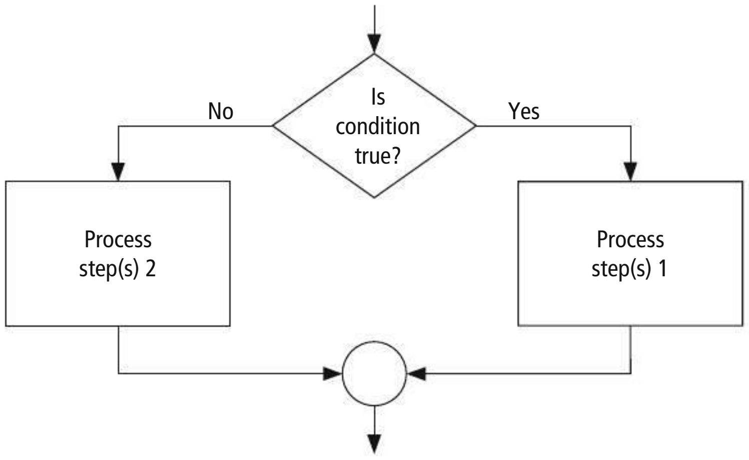


FIGURE 1.10 **Pseudocode and flowchart for a decision structure.**

executed. If the answer is "no", then another is executed. This structure is called a **decision structure**. Figure 1.10 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".

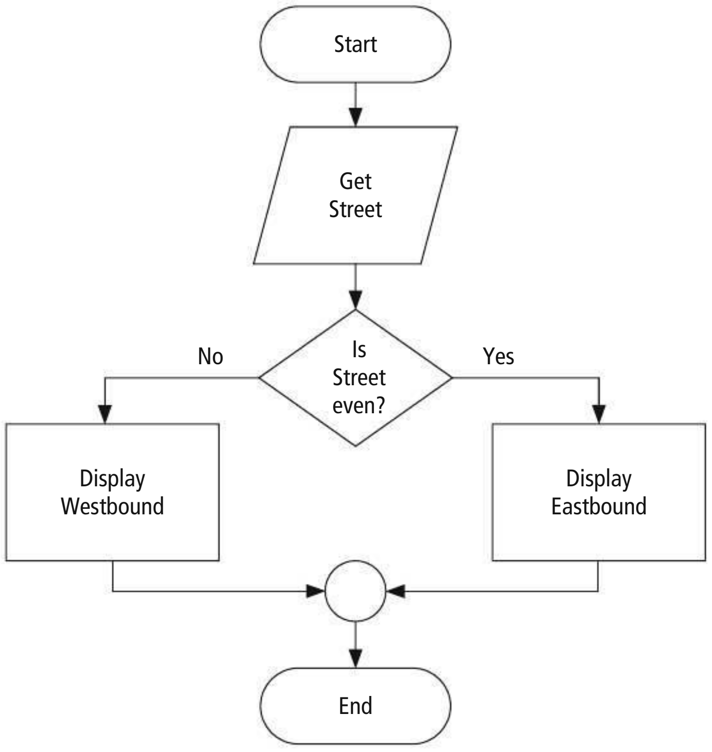


FIGURE 1.11 **Flowchart for the numbered New York City streets problem.**

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

FIGURE 1.12 **Pseudocode for the numbered New York City streets problem.**

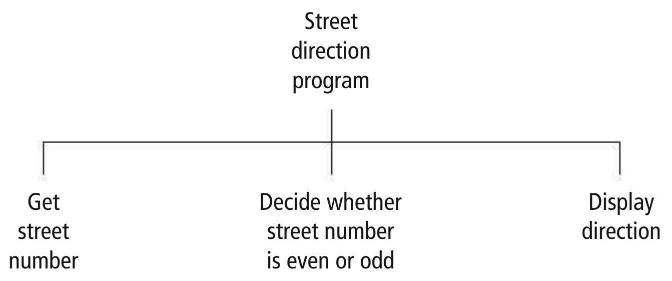


FIGURE 1.13 **Hierarchy chart for the numbered New York City streets problem.**

Figures 1.11 through 1.13 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. 1.14. The solution of the next problem requires a repetition structure.

```
Do While condition is true
  Process step(s)
Loop
```

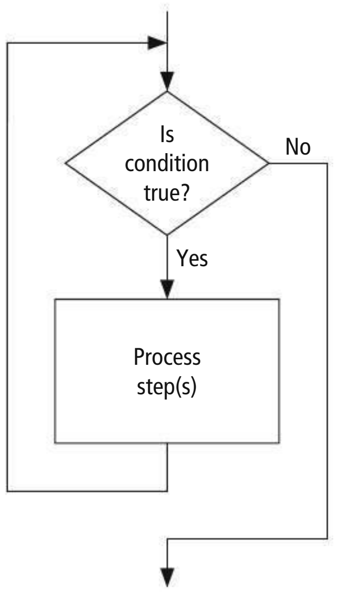


FIGURE 1.14 **Pseudocode and flowchart for a loop.**

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. 1.15 to 1.17.

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.

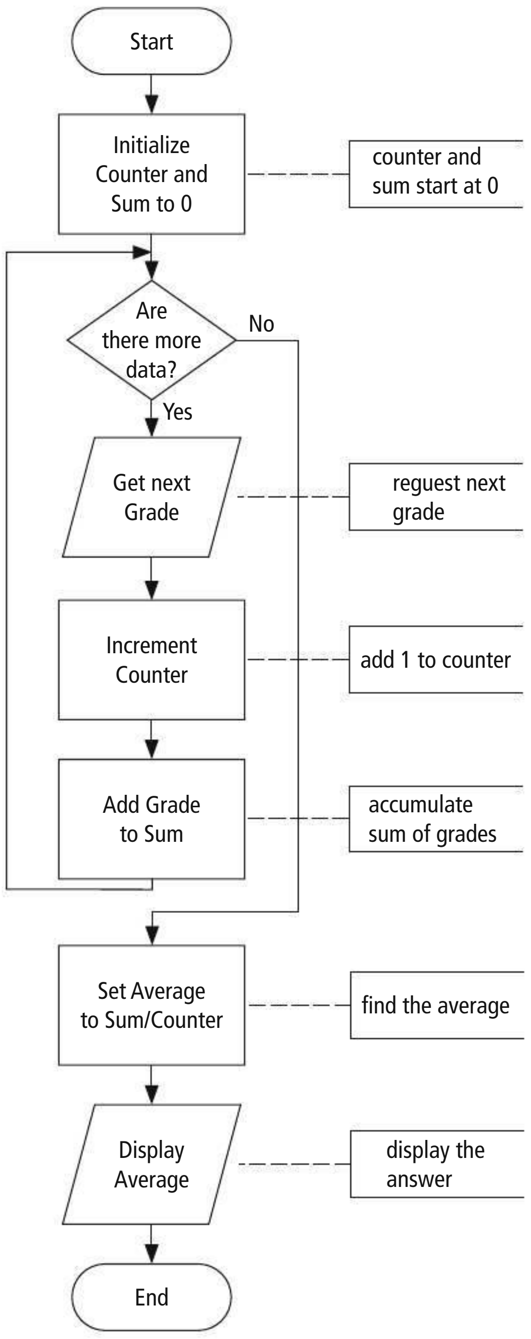


FIGURE 1.15 **Flowchart for the class average problem.**

```
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 1.16 **Pseudocode for the class average problem.**

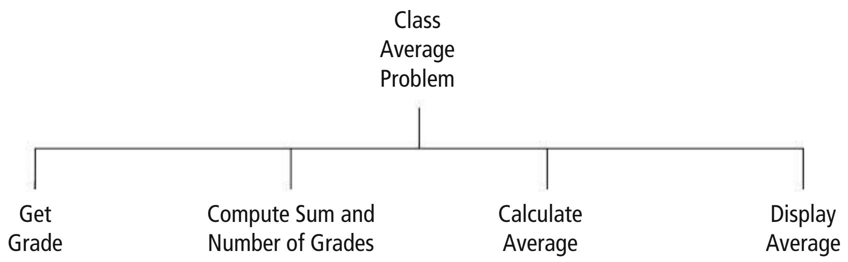


FIGURE 1.17 **Hierarchy chart for the class average problem.**

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, 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 2017

Visual Basic 2017 is a recent version 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.

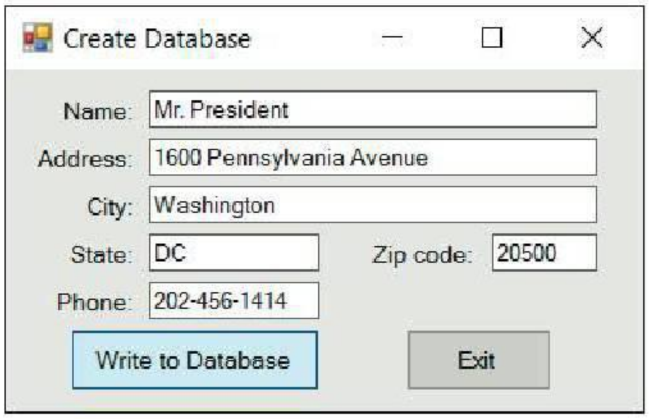
Why Windows and Why Visual Basic?

What people call **graphical user interfaces**, or GUIs, 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!



Name: Mr. President
Address: 1600 Pennsylvania Avenue
City: Washington
State: DC
Zip code: 20500
Phone: 202-456-1414

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.

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 10) or Windows Explorer (with Windows 7) to create a folder to hold your programs.

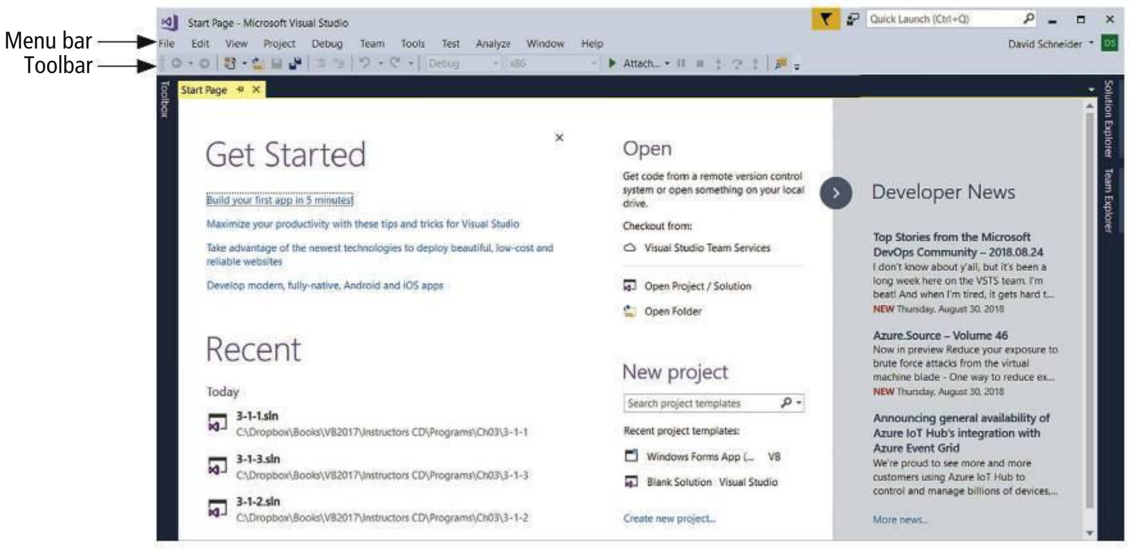
The process for starting Visual Basic varies slightly with the version of Windows and the edition of Visual Studio installed on the computer. Some possible sequences of steps are shown below.

Windows 7 Click the Windows *Start* button, click *All Programs*, and then click on "Microsoft Visual Studio 2017."

Windows 8 Click the tile labeled "Visual Studio 2017." If there is no such tile, click on *Search* in the Charms bar, select the Apps category, type "Visual Studio" into the Search box in the upper-right part of the screen, and click on the rectangle labeled "Visual Studio 2017" that appears on the left side of the screen.

Windows 10 Click on the tile or icon labeled *Visual Studio 2017*.

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.



Get Started

Build your first app in 5 minutes!

Maximize your productivity with these tips and tricks for Visual Studio

Take advantage of the newest technologies to deploy beautiful, low-cost and reliable websites

Develop modern, fully-native, Android and iOS apps

Recent

Today

3-1-1.sln
C:\Dropbox\Books\VB2017\Instructors CD\Programs\Ch03\3-1-1

3-1-3.sln
C:\Dropbox\Books\VB2017\Instructors CD\Programs\Ch03\3-1-3

3-1-2.sln
C:\Dropbox\Books\VB2017\Instructors CD\Programs\Ch03\3-1-2

Open

Get code from a remote version control system or open something on your local drive.

Checkout from:

Visual Studio Team Services

Open Project / Solution

Open Folder

New project

Search project templates

Recent project templates:

Windows Forms App (... VB

Blank Solution Visual Studio

Create new project...

Developer News

Top Stories from the Microsoft DevOps Community – 2018.08.24
I don't know about y'all, but it's been a long week here on the VSTS team. I'm beat! And when I'm tired, it gets hard t...
NEW Thursday, August 30, 2018

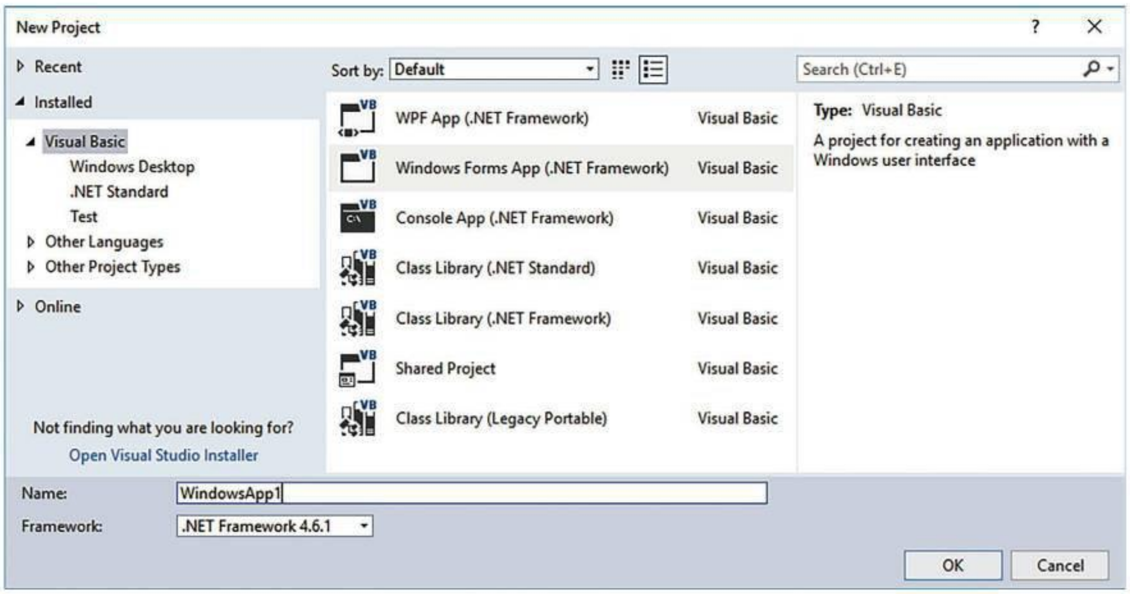
Azure.Source – Volume 46
Now in preview Reduce your exposure to brute force attacks from the virtual machine blade - One way to reduce ex...
NEW Thursday, August 30, 2018

Announcing general availability of Azure IoT Hub's integration with Azure Event Grid
We're proud to see more and more customers using Azure IoT Hub to control and manage billions of devices,...

More news...

FIGURE 2.3 **Visual Basic opening screen.**

The first item on the Menu bar is File. Click on File and then click on *New Project* in the dropdown menu to produce a New Project dialog box [Alternately, click on the New Project button ( ) on the Toolbar.] Figure 2.4 shows a New Project dialog box produced by the Visual Basic Community 2017 edition. Your screen might look slightly different than Fig. 2.4 even if you are using the Visual Basic Community 2017 edition.



Sort by: Default

WPF App (.NET Framework)	Visual Basic
Windows Forms App (.NET Framework)	Visual Basic
Console App (.NET Framework)	Visual Basic
Class Library (.NET Standard)	Visual Basic
Class Library (.NET Framework)	Visual Basic
Shared Project	Visual Basic
Class Library (Legacy Portable)	Visual Basic

Type: Visual Basic

A project for creating an application with a Windows user interface

Name: WindowsApp1

Framework: .NET Framework 4.6.1

FIGURE 2.4 **The Visual Basic New Project dialog box.**

Select *Visual Basic* in the list on the left side of Fig. 2.4, and select *Windows Forms App* (.NET Framework) 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 WindowsApp1, can be specified at this time. Since we will have a chance to change it later, let's just use the name WindowsApp1 for 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**. The IDE contains the tools for writing, running, and debugging programs.

Your screen might 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 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 vertical 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 Tools, 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**.

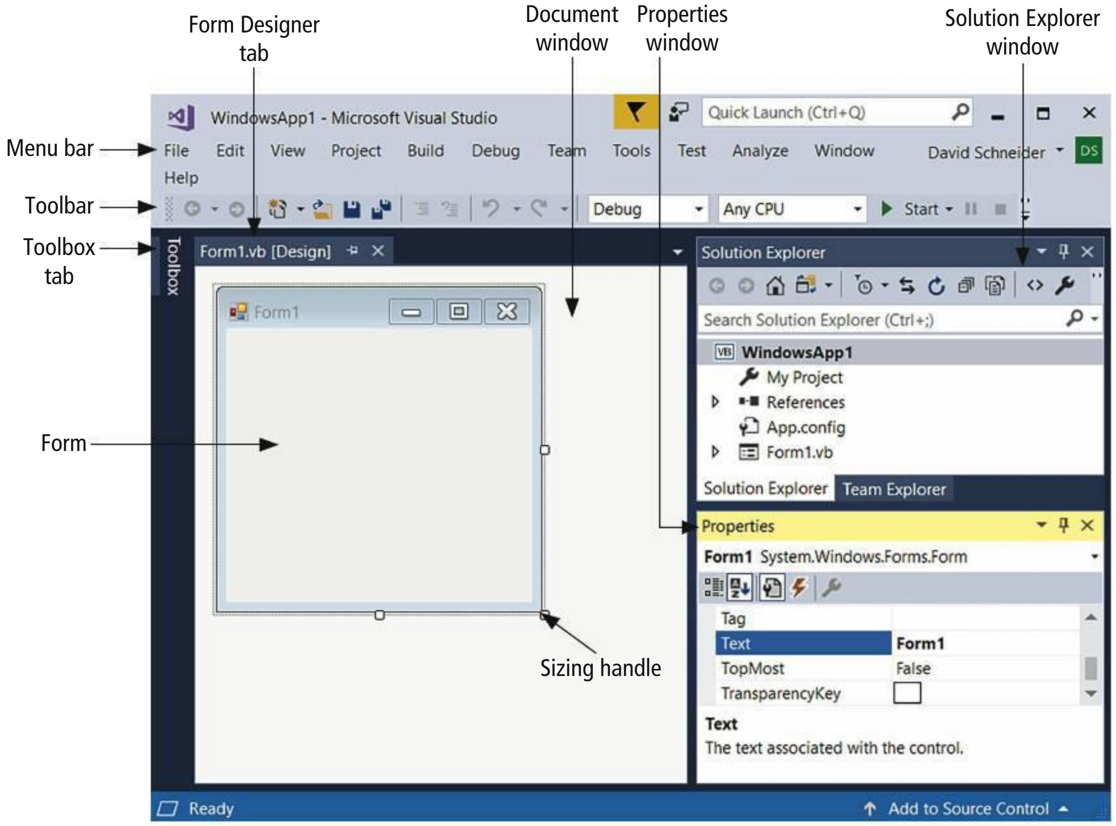


FIGURE 2.5 **The Visual Basic Integrated Development Environment in Form Designer mode.**

The **Document window** currently holds the rectangular **Form window**, or **form** for short. (The Form window is also known as the **form designer window** or the **design window**.) 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 its sizing handles.

The **Properties window** is used to change the initial appearance and behavior of objects on the form. Some (but not all) properties and appearances can be changed by code.

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, click on the Toolbox tab at the left side of the screen. The Toolbox will come into view. Then click on the push-pin 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.

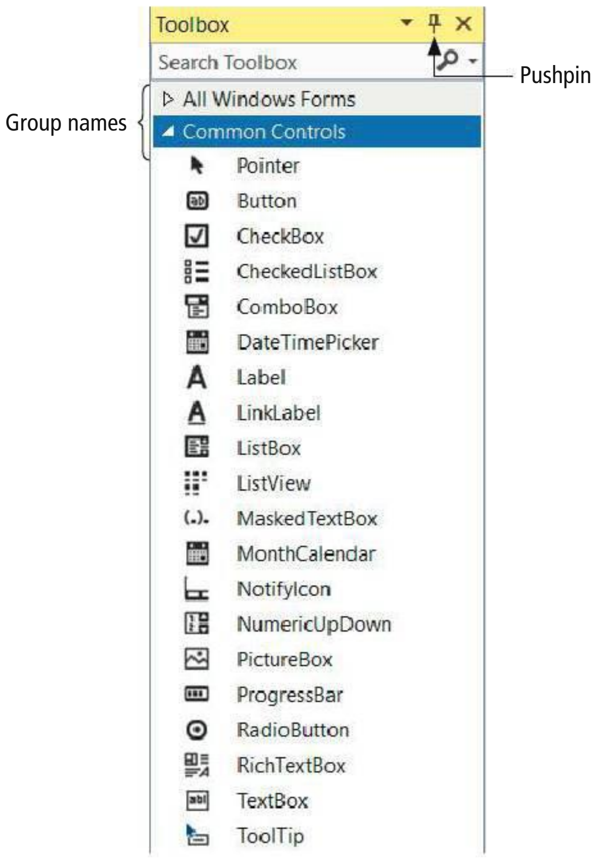


FIGURE 2.6 **The Toolbox's common controls.**

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.

VideoNote
Textbox Walkthrough

A Text Box Walkthrough

Place a text box on a form

1. Click on *New Project* in the File menu and begin a new Visual Basic program.
2. Double-click on the TextBox control (abl 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.

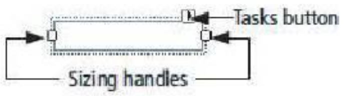


FIGURE 2.7 **Setting the Text property.**

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.

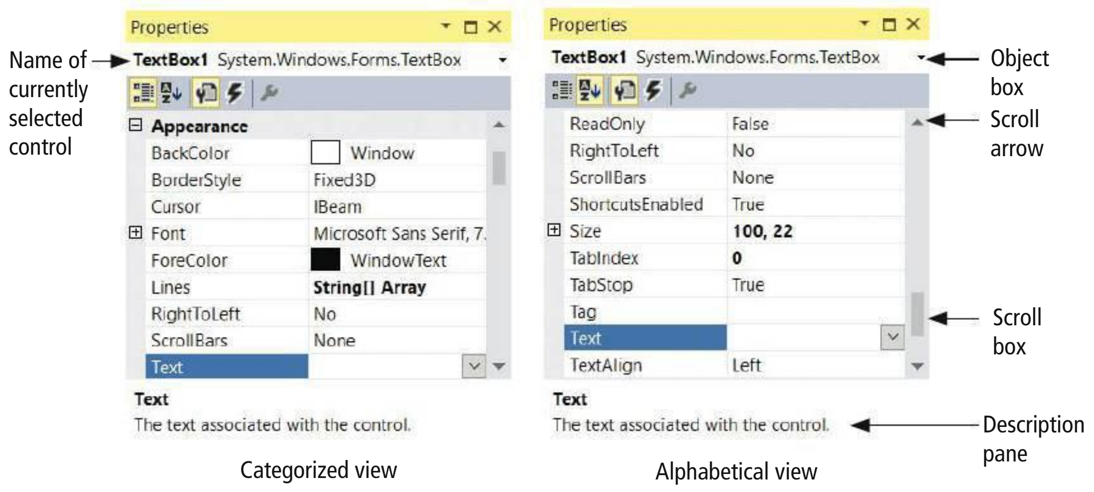
   You can now alter the width and location as before. ***Note:*** The text box should now be selected. If not, click anywhere inside the text box to select it.

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.



TextBox1 System.Windows.Forms.TextBox

⊟ **Appearance**	
BackColor	☐ Window
BorderStyle	Fixed3D
Cursor	IBeam
⊞ Font	Microsoft Sans Serif, 7.
ForeColor	■ WindowText
Lines	**String[] Array**
RightToLeft	No
ScrollBars	None
Text	

Text
The text associated with the control.

TextBox1 System.Windows.Forms.TextBox

ReadOnly	False
RightToLeft	No
ScrollBars	None
ShortcutsEnabled	True
⊞ Size	**100, 22**
TabIndex	**0**
TabStop	True
Tag	
Text	
TextAlign	Left

Text
The text associated with the control.

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 **free-floating** or **undocked**. Some people find a free-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 will now 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.

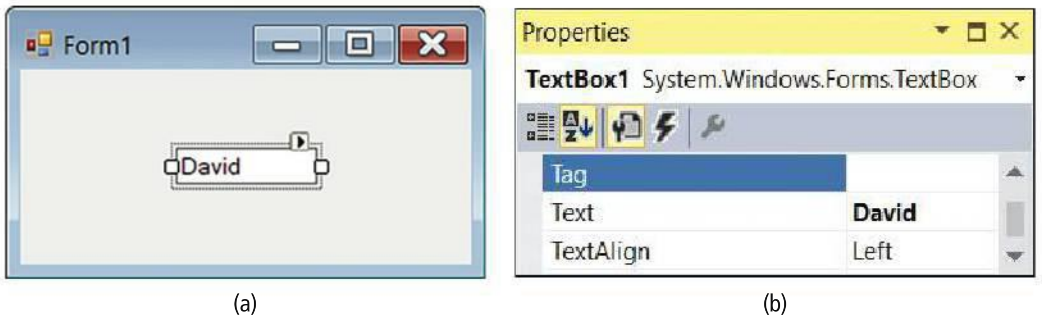
Note 1: The third and fourth buttons below the Object box, the *Properties* button and the *Events* button, determine whether properties or events are displayed in the Properties window. If the *Properties* button is not highlighted, click on it.

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.



Tag
Text David
TextAlign Left

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.

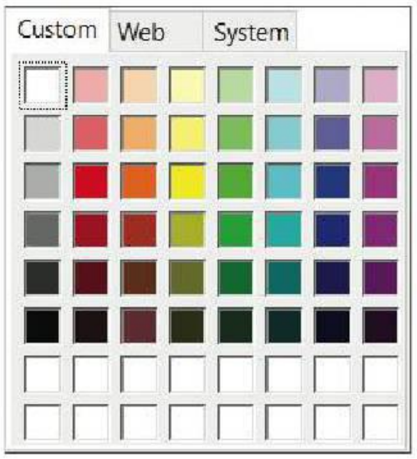


FIGURE 2.10 **Setting the ForeColor property.**

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.

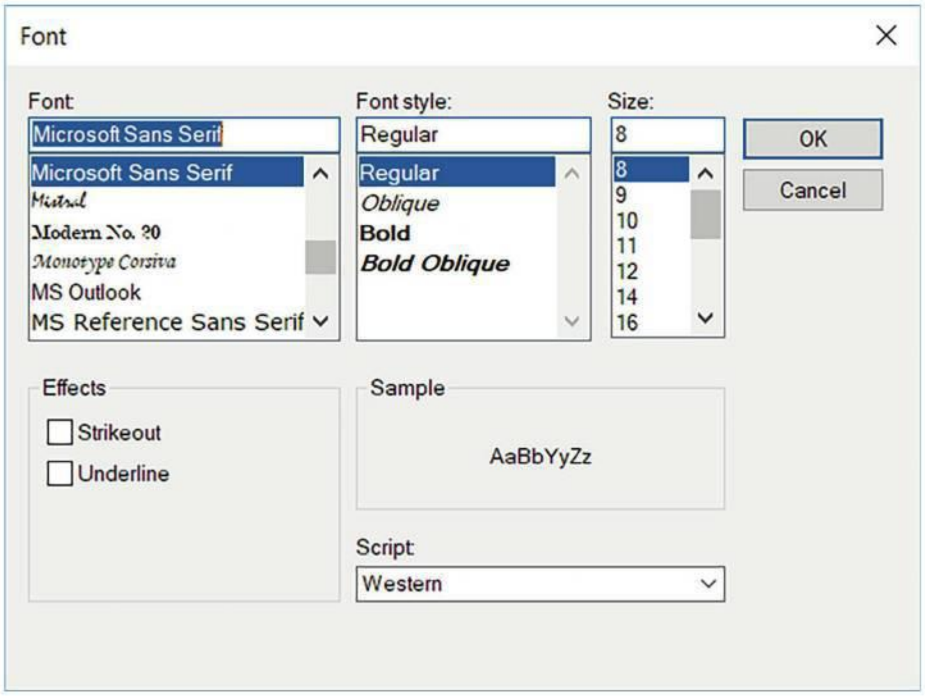


FIGURE 2.11 **The Font dialog box.**

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, and the text box expanded to 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 (▶ Start ▾) 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 red *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 using the shortcut key combination next to *Stop Debugging* in the Debug menu.

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.

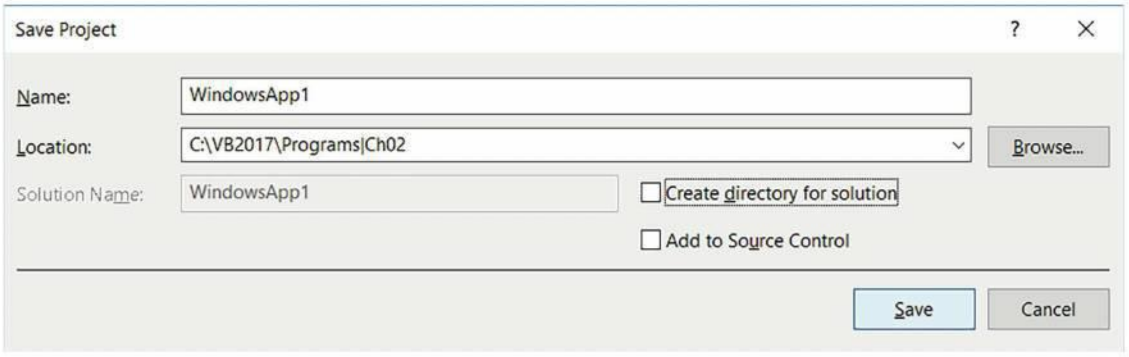
Such a text box is used for output. Only code can display information in the text box. (***Note:*** In this textbook, whenever a text box will be used only for the purpose of displaying output, we will always set the ReadOnly property to True.)

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.



Save Project

Name: WindowsApp1

Location: C:\VB2017\Programs|Ch02

Solution Name: WindowsApp1

☐ Create directory for solution

☐ Add to Source Control

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 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 the 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. (***Note:*** You can also carry out the task in the first sentence by using the shortcut key combination next to *Open Project* in the File menu.)

34. Click on *Close Project* in the File menu to close the program.

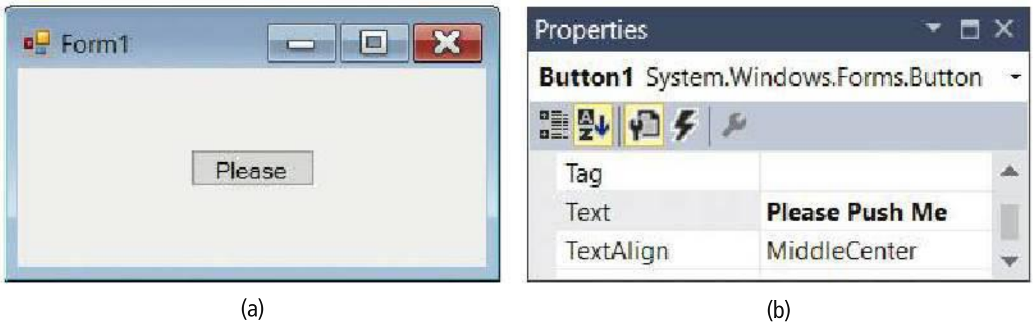
A Button Walkthrough

VideoNote
Button Walkthrough

Place a button on a form

1. Click on the *New Project* button ( ) on the Toolbar and begin a new program.
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.



Properties
Button1 System.Windows.Forms.Button

Tag	
Text	**Please Push Me**
TextAlign	MiddleCenter

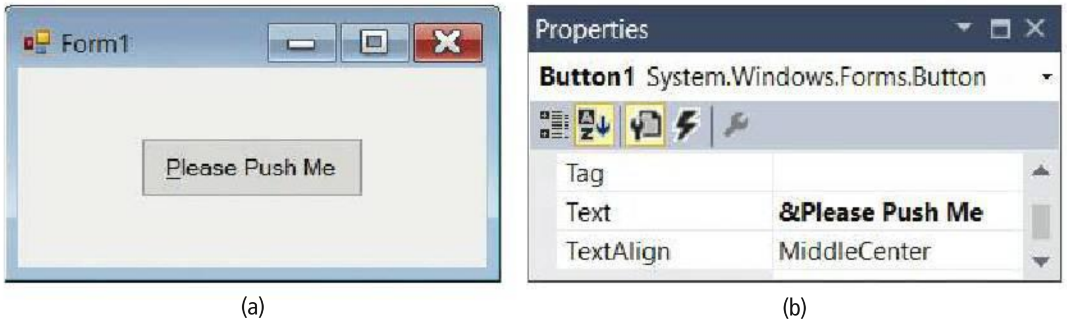
FIGURE 2.13 **Setting the Text property.**

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 turns blue 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 code to be executed 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.)



Properties
Button1 System.Windows.Forms.Button

Tag	
Text	**&Please Push Me**
TextAlign	MiddleCenter

FIGURE 2.14 **Designating P as an access key.**

9. Click on *Close Project* in the File menu to close the program.

   There is no need to save this program, so click on the *Discard* button in the message box that appears.

A Label Walkthrough

1. Click on the *New Project* button on the Toolbar and begin a new program.

   Feel free to keep the default name, such as WindowsApp1.
2. Double-click on the Label control (A 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 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 on the Toolbar and begin a new program.

   Feel free to keep the default name, such as WindowsApp1.
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 Project* in the File menu to close the program.

   There is no need to save this program, so click on the *Discard* button in the message box that appears.

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.

Both the Name and Text properties of a button are initially set to something like Button1. However, changing one of these properties does not affect the setting of the other properties, 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 enabled when the pushpin icon is horizontal (⊣). When the Auto Hide feature is enabled, the Toolbox window will move 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. Click on the vertical Toolbox tab to display the Toolbox, and then move the mouse cursor somewhere outside the Toolbox window and click the left mouse button.

   The window becomes a tab captioned Toolbox on the left side of the screen.
4. Click on the tab.

   The window comes 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.

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.

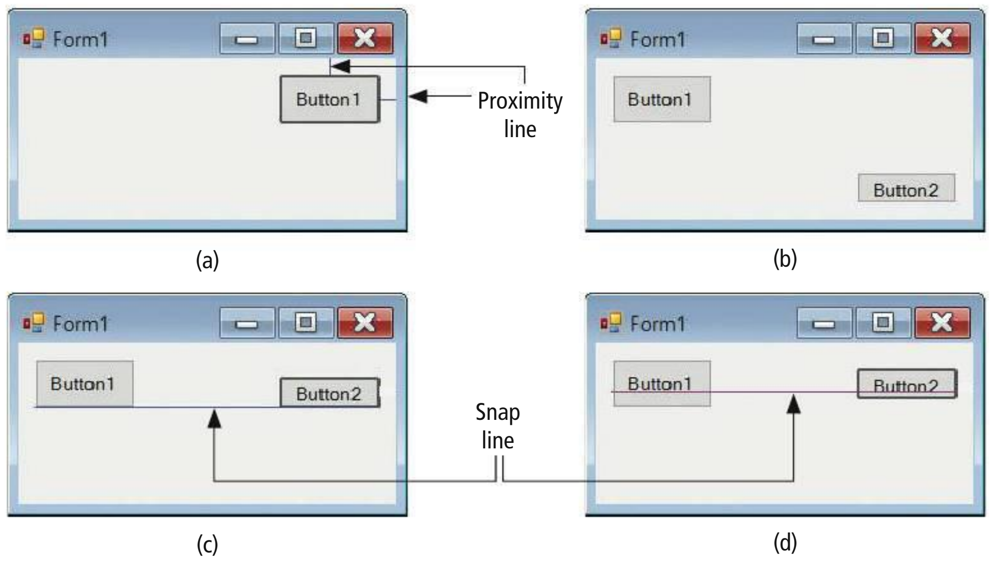


FIGURE 2.15 **Positioning controls.**

4. Place a second button directly 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 texts in 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. Position the two buttons as shown in Fig.15 (b). 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: Adjust the vertical spacing between two or more selected controls.

   Horizontal Spacing: Adjust the horizontal spacing between two or more selected controls.
10. With the two buttons still selected, activate 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 **dominant control** of the group) will have white sizing handles, while the other controls will have black sizing handles. All alignment and sizing instructions initiated from the Format menu will keep the dominant control fixed and will align (or size) the other controls with respect to the dominant control. You can designate a different control to be the dominant 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.

Setting Tab Order

During the execution of a Program, only one control can receive user input through the keyboard. That control is said to have the ***focus***. When a text box has the focus, there is a blinking cursor inside it.

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 the form.
3. Run the program, and then 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**.

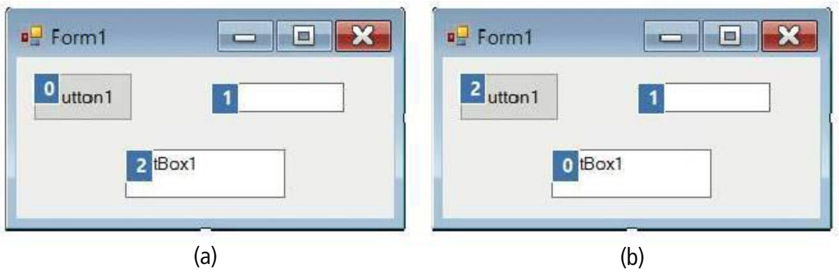


FIGURE 2.16 **Tab order.**

6. Click on the list box, then the text box, and finally the button.

   Notice that the tab indexes change as shown in Fig. 2.16(b).
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 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, cannot receive the focus, and cannot respond to the user. 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 drag the control from the Toolbox to the location in the form.

6. There is 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 a button to have a multi-line 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 the properties windows in Fig. 2.8. 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.

9. Exercises 35 through 47 develop additional techniques for manipulating and accessing controls placed on a form. We recommend that you work these exercises whether or not they are assigned by your instructor.

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 two buttons, run the program, and click on each button. What do you notice different about a button after it has been clicked?

2. While a program is running, a control is said to lose focus when the focus moves from that control to another control. Give three ways the user can cause a control to lose focus.