starting out with >>>

VISUAL BASIC

EIGHTH EDITION

TONY GADDIS · KIP IRVINE

STARTING OUT WITH

Visual Basic®

Eighth Edition

STARTING OUT WITH

Visual Basic®

Eighth Edition

Tony Gaddis
Haywood Community College

Kip Irvine
Florida International University

Pearson

Senior Vice President Courseware Portfolio Management: Marcia Horton
Vice President, Portfolio Management: Engineering, Computer Science & Global Editions: Julian Partridge
Executive Portfolio Manager: Matt Goldstein
Portfolio Management Assistant: Meghan Jacoby
Product Marketing Manager: Yvonne Vannatta
Field Marketing Manager: Demetrius Hall
Marketing Assistant: Jon Bryant
Managing Producer: Scott Disanno
Content Producer: Amanda Brands
Manufacturing Buyer, Higher Ed, Lake Side Communications, Inc. (LSC): Maura Zaldivar-Garcia
Cover Designer: Pearson CSC
Manager, Rights and Permissions: Ben Ferrini
Cover Image: matin/shutterstock
Inventory Manager: Bruce Boundy
Full-Service Project Manager: Abhishan Sharma, Integra Software Services Pvt. Ltd.
Composition: Integra Software Services Pvt. Ltd.
Printer/Binder: Lake Side Communications, Inc.

Credits and acknowledgments borrowed from other sources and reproduced, with permission, in this textbook appear on the Credits page at the end of the text.

Library of Congress Cataloging-in-Publication Data
Names: Gaddis, Tony, author. | Irvine, Kip R., author.
Title: Starting out with visual basic / Tony Gaddis, Haywood Community College, Kip Irvine, Florida International University.
Description: Eighth edition. | Hoboken : Pearson, [2020] | Includes bibliographical references and index.
Identifiers: LCCN 2018048219 | ISBN 9780135204658 | ISBN 0135204658
Subjects: LCSH: Visual Basic (Computer program language)
Classification: LCC QA76.73.M53 G33 2019 | DDC 005.26/8—dc23 LC record available at https://lccn.loc.gov/2018048219

10 9 8 7 6 5 4 3 2 1

Pearson

ISBN 10: 0-13-520465-8
ISBN 13: 978-0-13-520465-8

Contents in Brief

Contents

VideoNote

LOCATIONS OF VIDEONOTES IN THE TEXT

Chapter 6	Creating and calling procedures, p. 383
	Passing Arguments to Procedures, p. 387
	Functions, p. 395
	The *Sale Price Calculator* application, p. 396
	Practicing the *Step Into* command, p. 404
	Improving the *Kayak Rental* Application, p. 417
	The *Retail Price Calculator* Problem, p. 420
Chapter 7	Creating and Displaying a Second Form, p. 431
	Creating an application with two forms, p. 437
	Creating a Menu, p. 461
	Adding menus and forms to the *Kayak Rental* Application, p. 482
	The *Astronomy Helper* Problem, p. 497
Chapter 8	Accessing Array Elements with a Loop, p. 502
	Using an array to hold a list of names entered by the user, p. 507
	Using array elements in a calculation, p. 510
	Using Arrays to Look Up Information in the *Kayak Rental* Application, p. 561
	Finding the largest and smallest values in an array, p. 566
	Adding students to clubs, p. 569
	The Lottery Simulation, p. 571
Chapter 9	Writing Data to a File, p. 575
	Reading Data from a File, p. 581
	Detecting the end of a file, p. 588
	Files and Colors in the *Kayak Rental* Application, p. 619
Chapter 10	The DataGridView Control, p. 636
	Data-Bound Controls, p. 648
	Binding individual controls to the *SalesStaff* table, p. 655
	Adding a Database to the *Kayak Rental* Application, p. 703
	The *Kayak Browser* Problem, p. 706
	The *Karate Payments by a Single Member* Problem, p. 709
Chapter 11	Creating a Simple Web Application, p. 716
	The *Student Picnic* application, p. 726
	Building a Movie Tracking Application, p. 762
	The *Stadium* Seating Problem, p. 765
Chapter 12	Creating a Class, p. 771
	Creating and using a Collection, p. 788
	Completing the Student *Collection* application, p. 794
	Kayak Rental Application with Classes, p. 817
	The *Motor Class* Problem, p. 825

Preface

Welcome to *Starting Out with Visual Basic*, Eighth Edition. This edition has been revised and updated for Visual Studio 2017. It is intended for use in an introductory programming course. It is designed for students who have no prior programming background, but even experienced students will benefit from its depth of detail and the chapters covering databases, Web applications, and other advanced topics. The book is written in clear, easy-to-understand language and covers all the necessary topics of an introductory programming course. The text is rich in concise, practical, and real-world example programs, so the student not only learns how to use the various controls, constructs, and features of Visual Basic, but also learns why and when to use them.

A Look at Visual Basic: Past and Present

The first version of Visual Basic was introduced in 1991. Prior to its introduction, writing a GUI interface for an application was no small task. Typically, it required hundreds of lines of C code for even the simplest *Hello World* program. Additionally, an understanding of graphics, memory, and complex system calls was often necessary. Visual Basic was revolutionary because it significantly simplified this process. With Visual Basic, a programmer could visually design an application's user interface. Visual Basic would then generate the code necessary to display and operate the interface. This allowed the programmer to spend less time writing GUI code and more time writing code to perform meaningful tasks.

The evolution of Visual Basic from version 1 to version 6 followed a natural progression. Each new release was an improved version of the previous release, providing additional features and enhancements. Visual Basic versions offered backward compatibility, where code written in an older version was compatible with a newer version of the Visual Basic development environment.

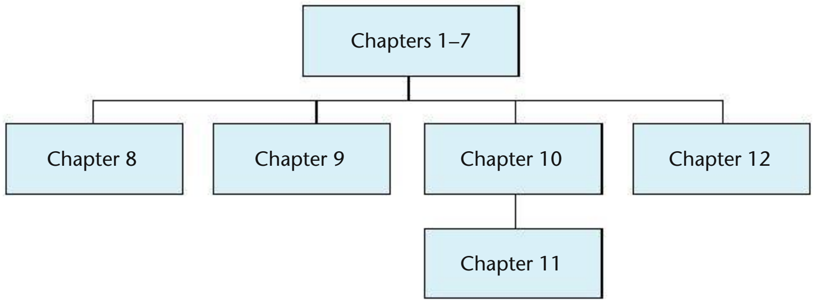
In 2002, Microsoft released a new object-oriented software platform known as .NET. The .NET platform consists of several layers of software that sit above the operating system and provide a secure, managed environment in which programs can execute. In addition to providing a managed environment for applications to run, .NET also provided new technologies for creating Internet-based programs and programs that provide services over the Web. Along with the introduction of the .NET platform, Microsoft introduced a new version of Visual Basic known as VB .NET 2002, which allowed programmers to write desktop applications or Web applications for the .NET platform.

VB .NET was not merely a new and improved version of VB 6, however. VB .NET was a totally new programming environment, and the Visual Basic language was dramatically revised. The changes were substantial enough that programs written in earlier versions of Visual Basic were not compatible with VB .NET. Microsoft provided a utility that could be used to convert older Visual Basic applications to the new VB .NET syntax, but the results were not always perfect. Although this was frustrating for some Visual Basic developers, Microsoft reasoned the changes were necessary to ensure that Visual Basic continued to evolve as a modern, professional programming environment.

Microsoft has continued to enhance and improve Visual Basic by regularly releasing new versions. The versions, which are named after the year in which they were released, are Visual Basic 2003, Visual Basic 2005, Visual Basic 2008 and so forth. At the time this book was written, the current release was Visual Basic 2017. You can see a complete list of the enhancements that have been made to this version of Visual Basic, as well as past versions, on this web page: `msdn.microsoft.com/en-us/library/we86c8x2.aspx`.

Organization of the Text

The text teaches Visual Basic step-by-step. Each chapter covers a major set of programming topics, introduces controls and GUI elements, and builds knowledge as the student progresses through the book. Although the chapters can be easily taught in their existing sequence, there is some flexibility. The following diagram suggests possible sequences of instruction.



Chapters 1 through 7 cover the fundamentals of program design, flow control, modular programming, and the most important Visual Basic controls. The instructor may then continue in any order with Chapters 8, 9, 10, or 12. Part of Chapter 11 relies on database concepts, so it should be covered after Chapter 10.

Brief Overview of Each Chapter

Chapter 1: Introduction to Programming and Visual Basic. This chapter provides an introduction to programming, the programming process, and Visual Basic. GUI programming and the event-driven model are explained. The components of programs, such as keywords, variables, operators, and punctuation are covered, and tools such as flowcharts and pseudocode are presented. The student gets started using the Visual Basic environment in a hands-on tutorial.

Chapter 2: Creating Applications with Visual Basic. In this chapter the student learns to create forms with labels, buttons, and picture boxes and learns to modify control properties. The student is introduced to Visual Basic code, and learns to write simple event-driven applications that respond to button clicks, or provide interaction through clickable images. This chapter introduces the *Visual Studio Help* system, and provides a tutorial on simple debugging. The importance of commenting code is also discussed.

Chapter 3: Variables and Calculations. Variables, constants, and the Visual Basic data types are introduced. The student learns to gather input and create simple arithmetic statements. The intricacies of GUI design are introduced as the student learns about grouping controls with group boxes, assigning keyboard access keys, and setting the tab order. The student is introduced to exceptions and learns to write simple exception handlers. Debugging techniques for locating logic errors are covered.

Chapter 4: Making Decisions. The student learns about relational operators and how to control the flow of a program with the `If... Then`, `If...Then... Else`, and `If... Then... ElseIf` statements. Logical operators are introduced, and the `Select Case` statement is covered. Important applications of these constructs are discussed, such as testing numeric values, strings, and determining if a value lies within a range, and validating user input. Several string-handling functions and string methods are discussed. Radio buttons and check boxes are also introduced.

Chapter 5: Lists and Loops. This chapter begins by showing the student how to use input boxes as a quick and simple way to gather input. Next, list boxes and combo boxes are introduced. The chapter covers repetition control structures: the `Do While`, `Do Until`, and `For... Next` loops. Counters, accumulators, running totals, and other loop-related topics are discussed. The student also learns how to generate random numbers.

Chapter 6: Procedures and Functions. The student learns how and why to modularize programs with general-purpose procedures and functions. Arguments, parameters, and return values are discussed. Debugging techniques for stepping into and over procedures are introduced.

Chapter 7: Multiple Forms, Modules, and Menus. This chapter shows how to add multiple forms to a project and how to create a module to hold procedures and functions that are not associated with a specific form. It covers creating a menu system, with commands and submenus that the user may select from.

Chapter 8: Arrays and More. This chapter discusses both single dimension and multidimensional variable arrays. Many array programming techniques are presented, such as summing all the elements in an array, summing all the rows or columns in a two-dimensional array, searching an array for a specific value, sorting arrays, and using parallel arrays. The Enabled property, timer controls, and control anchoring and docking are also covered.

Chapter 9: Files, Printing, and Structures. This chapter begins by discussing how to save data to sequential text files and then read the data back into an application. The OpenFileDialog, SaveFileDialog, FontDialog, and ColorDialog controls are introduced. The PrintDocument control is discussed, with a special focus on printing reports. The chapter shows the student how to create user-defined data types with structures.

Chapter 10: Working with Databases. This chapter introduces basic database concepts. The student learns how to display a database table in a DataGridView control and write applications that display, sort, and update database data. The Structured Query Language (SQL) is introduced. An application that shows how to display database data in list boxes, text boxes, labels, and combo box is presented. The chapter concludes with an overview of Language Integrated Query (LINQ).

Chapter 11: Developing Web Applications. This chapter shows the student how to create ASP.NET applications that run on Web Browsers such as Internet Explorer, Chrome, Firefox, and Safari. Using Microsoft Visual Studio, or Microsoft Visual Web Developer, the student learns how to use Web server controls and Web forms to build interactive, database-driven Web applications.

Chapter 12: Classes, Collections, and Inheritance. This chapter introduces classes as a tool for creating abstract data types. The process of analyzing a problem and determining its classes is discussed, and techniques for creating objects, properties, and methods are introduced. Collections are presented as structures for holding groups of objects. The *Object Browser*, which allows the student to see information about the classes, properties, methods, and events available to a project, is also covered. The chapter concludes by introducing inheritance, and shows how to create a class that is based on an existing class.

Appendix A: Advanced User Interface Controls and Techniques. Discusses many of the more advanced controls available in Visual Basic, as well as several helpful programming techniques. This appendix also provides a summary of common user interface design guidelines.

Appendix B: Windows Presentation Foundation (WPF). Introduces the student to the Windows Presentation Framework (WPF), and includes a tutorial in which the student creates a simple WPF application.

Appendix C: Converting Mathematical Expressions to Programming Statements. Shows the student how to convert a mathematical expression into a Visual Basic programming statement.

Appendix D: Answers to Checkpoints. Students may test their progress by comparing their answers to Checkpoints with the answers provided. The answers to all Checkpoints are included.

Appendix E: Glossary. Provides a glossary of the key terms presented in the text.

The following appendixes can be downloaded from the book's companion Web site at `www.pearson.com/gaddis`.

Appendix F: Visual Basic Function and Method Reference. Provides a reference for the functions and methods that are covered in the text. The exceptions that may be caused by these functions and methods are also listed.

Appendix G: Binary and Random-Access Files. Describes programming techniques for creating and working with binary and random-access data files.

Features of the Text

Concept Statements. Each major section of the text starts with a concept statement. This statement concisely summarizes the meaning of the section.

Tutorials. Each chapter has several hands-on tutorials that reinforce the chapter's topics. Many of these tutorials involve the student in writing applications that can be applied to real-world problems.

VideoNote

VideoNotes. A series of online videos, developed specifically for this book, are available for viewing at `http://www.pearson.com/gaddis`. Icons appear throughout the text alerting the student to videos about specific topics.

Checkpoints. Checkpoints are questions placed at intervals throughout each chapter. They are designed to query the student's knowledge immediately after learning a new topic. Answers to all the Checkpoints are provided in Appendix D.

Notes. Notes are short explanations of interesting or often misunderstood points relevant to the topic being discussed.

Tips. Tips advise the student on the best techniques for approaching different programming problems and appear regularly throughout the text.

Warnings. Warnings caution the student about certain Visual Basic features, programming techniques, or practices that can lead to malfunctioning programs or lost data.

Review Questions and Exercises. In the tradition of all Gaddis texts, each chapter presents a thorough and diverse set of review questions and exercises. These include traditional fill-in-the-blank, true or false, multiple choice, and short answer questions. There are also unique tools for assessing a student's knowledge. For example, *Find the Error* questions ask the student to identify syntax or logic errors in brief code segments. *Algorithm Workbench* questions ask the student to design code segments to satisfy a given problem. There are also *What Do You Think?* questions that require the student to think critically and contemplate the topics presented in the chapter.

Programming Challenges. Each chapter offers a pool of programming exercises designed to solidify the student's knowledge of the topics at hand. In most cases, the assignments present real-world problems to be solved. When applicable, these exercises also include input validation rules.

Supplements

Student

The following supplementary material is bundled with the book:

- Source code and files required for the chapter tutorials are available at the book's companion website: `www.pearson.com/gaddis`.
- The website also contains Appendix F, *Visual Basic Function and Method Reference*, and Appendix G, *Binary and Random-Access Files*.

Instructor

The following supplements are available to qualified instructors:

- Answers to all Review Questions in the text
- Solutions for all Programming Challenges in the text
- PowerPoint presentation slides for every chapter
- Test bank
- Test generation software that allows instructors to create customized tests

For information on how to access these supplements, visit the Pearson Education Instructor Resource Center at `http://www.pearson.com`.

Online Practice and Assessment with MyLab Programming

MyLab Programming is a web-based service that 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 beginning students who often 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 small 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. 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, visit `www.pearson.com/mylab/programming`.

Web Resources

Self-assessment quizzes, PowerPoint slides, source code files, and glossary flashcards are available on the Companion Website for *Starting Out with Visual Basic* at `www.pearson.com/gaddis`.

Acknowledgments

There were many helping hands in the development and publication of this text. The authors would like to thank the following faculty reviewers for their helpful suggestions and expertise:

Achla Agarwal, *Bossier Parrish Community College*
Ronald Bass, *Austin Community College*
Ronald Beauchemin, *Springfield Technical Community College*
Zachory T. Beers, *Microsoft Corporation*
Robert M. Benavides, *Collin County Community College District*
Paul T. Bladek, *Edmonds Community College*
Douglas Bock, *Southern Illinois University at Edwardsville*
Skip Bottom, *J. Sargeant Reynolds Community College*
Harold Broberg, *Indiana Purdue University*
Nancy Burns, *Professor of Computer Science, Chipola College*
Arthur E. Carter, *Radford University*
Mara Casado, *State College of Florida, Manatee-Sarasota*
Joni Catanzaro, *Louisiana State University*
Jesse Cecil, *College of the Siskiyous*
Dr. Robert Coil, *Cincinnati State Community and Technical College*
Mohammad Dadashzadeh, *Oakland University*
Travis Dalton, *Columbia College (SC)*
Carol A. DesJardins, *St. Clair County Community College*
Sallie Dodson, *Radford University*
William J. Dorin, *Indiana University*
Robert Ekblaw, *SUNY Albany*
Rose M. Endres, *City College of San Francisco*
Jean Evans, *Brevard Community College*
Mark Fienup, *University of Northern Iowa*
Pierre M. Fiorini, PhD, *University of Southern Maine*
Arlene Flerchinger, *Chattanooga State Technical Community College*
Lawrence Fudella, *Erie Community College*
Gail M. Gehrig, *Florida Community College at Jacksonville*
Jayanta Ghosh, *Florida Community College*
Martha Gibson, *Central Texas College*
Iskandar Hack, *Indiana University—Purdue University at Fort Wayne*
Tom Higginbotham, *Southeastern Louisiana University*
Dennis Higgins, *SUNY Oneonta*
David M. Himes, *Oklahoma State University, Okmulgee*
Greg Hodge, *Northwestern Michigan College*
Corinne Hoisington, *Central Virginia Community College*
Jackie Horton, *University of Vermont*
May-Chuen Hsieh, *Southwest Tennessee Community College*
Lee A. Hunt, *Collin County Community College*
Darrel Karbginsky, *Chemeketa Community College*
Frank J. Kreimendahl, *University of New Hampshire*
Herb Kronholm, *Mid-State Technical College*
Phil Larschan, *Tulsa Community College*
Linda Lau, *Longwood University*
Art Lee, *Lord Fairfax Community College*
Joo Eng Lee-Partridge, *Central Connecticut State University*
Jing Liu, *Southeastern Louisiana University*
Juan Marquez, *Mesa Community College*
Gary Marrer, *Glendale Community College*
Norman McNeal, *Dakota County Technical College*
George McOuat, *Hawaii Pacific University*
Joseph Merrell
Sylvia Miner, *Florida International University*
Billy Morgan, *Holmes Community College*
Joan P. Mosey, *Point Park College*
Solomon Negash, *Kennesaw State University*
Robert Nields, *Cincinnati State Community and Technical College*
Gregory M. Ogle
Christopher J. Olson, *Dakota State University*

Merrill B. Parker, *Chattanooga State Technical Community College*
Rembert N. Parker, *Anderson University*
Alison Pechenick, *University of Vermont*
Richard Pelletier, *San Diego City College*
Carol M. Peterson, *South Plains Community College*
Anita Philipp, *Oklahoma City Community College*
T. N. Rajashekhara, *Camden County College*
Gregory Ramsay, *Morgan State University*
Mark Reis, *University of Virginia*
Malu Roldan, *San Jose State*
Pete Sanderson, *Otterbein University*
Judy Scholl, *Austin Community College*
Gurmukh Singh, *SUNY at Fredonia*
Anne Spalding, *Mesa State College*
Judith A. Stafford, *Tufts University*
JoAnne Strickland, *Solano Community College*
Angeline Surber, *Mesa Community College*
Robert L. Terrell, *Walters State Community College*
Joaquin Velez, *San Diego Community College District*
Margaret Warrick, *Allan Hancock College*
Doug Waterman, *Fox Valley Technical College*
Elaine Yale Weltz, *Seattle Pacific University*
Floyd Jay Winters, *Program Director, Computer Science, College of Florida, Manatee-Sarasota*
Catherine Wyman, *DeVry Institute, Phoenix*
Erik Wynters, *Bloomsburg University of Pennsylvania*
Sheri L. York, *Ball State University*
Youlong Zhuang, *Columbia College of Missouri*

The authors would like to thank their families for their tremendous support throughout this project. We would also like to thank everyone at Pearson who is part of the editorial, production, and marketing team. We are extremely fortunate to have Matt Goldstein as our editor. He has guided us through the delicate process of updating this book many times. We are also fortunate to have Demetrius Hall and Yvonne Vannatta as our marketing managers. Their hard work is truly inspiring, and they do a great job of getting this book out to the academic community. The production team, led by Amanda Brands, worked tirelessly to make this book a reality. Thanks to you all!

About the Authors

Tony Gaddis is the principal author of the *Starting Out with* series of textbooks. Tony has 20 years of experience teaching computer science courses at Haywood Community College in North Carolina. He is a highly acclaimed instructor who was previously selected as North Carolina's Community College *Teacher of the Year*, and has received the *Teaching Excellence* award from the National Institute for Staff and Organizational Development. Besides Visual Basic books, the *Starting Out with* series includes introductory books on programming logic and design, Alice, the C++ programming language, Java™, Python, Microsoft® Visual C#®, and MIT App Inventor, all published by Pearson.

Kip Irvine holds M.S. (computer science) and D.M.A. (music composition) degrees from the University of Miami. He was formerly on the faculty at Miami-Dade Community College, and is retired from the School of Computing and Information Sciences at Florida International University. His published textbooks include *COBOL for the IBM Personal Computer*, *Assembly Language for x86 Processors*, *C++ and Object-Oriented Programming*, and *Advanced Visual Basic .NET*.

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:

`http://www.pearson.com/gaddis`

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.

CHAPTER 1

Introduction to Programming and Visual Basic

TOPICS

Microsoft Visual Basic is a powerful software development system for creating applications that run on the Windows operating system. With Visual Basic, you can do the following:

- Create applications with graphical windows, dialog boxes, and menus
- Create applications that work with databases
- Create Web applications and applications that use Internet technologies
- Create applications that display graphics

Visual Basic, which is commonly referred to as VB, is a favorite tool among professional programmers. It provides tools to visually design an application's appearance, a modern programming language, and access to the latest Microsoft technologies. Powerful applications can be created with Visual Basic in a relatively short period of time.

Before plunging into learning Visual Basic, we will review the fundamentals of computer hardware and software, and then become familiar with the Visual Studio programming environment.

1.1 Computer Systems: Hardware and Software

CONCEPT: Computer systems consist of similar hardware devices and hardware components. This section provides an overview of computer hardware and software organization.

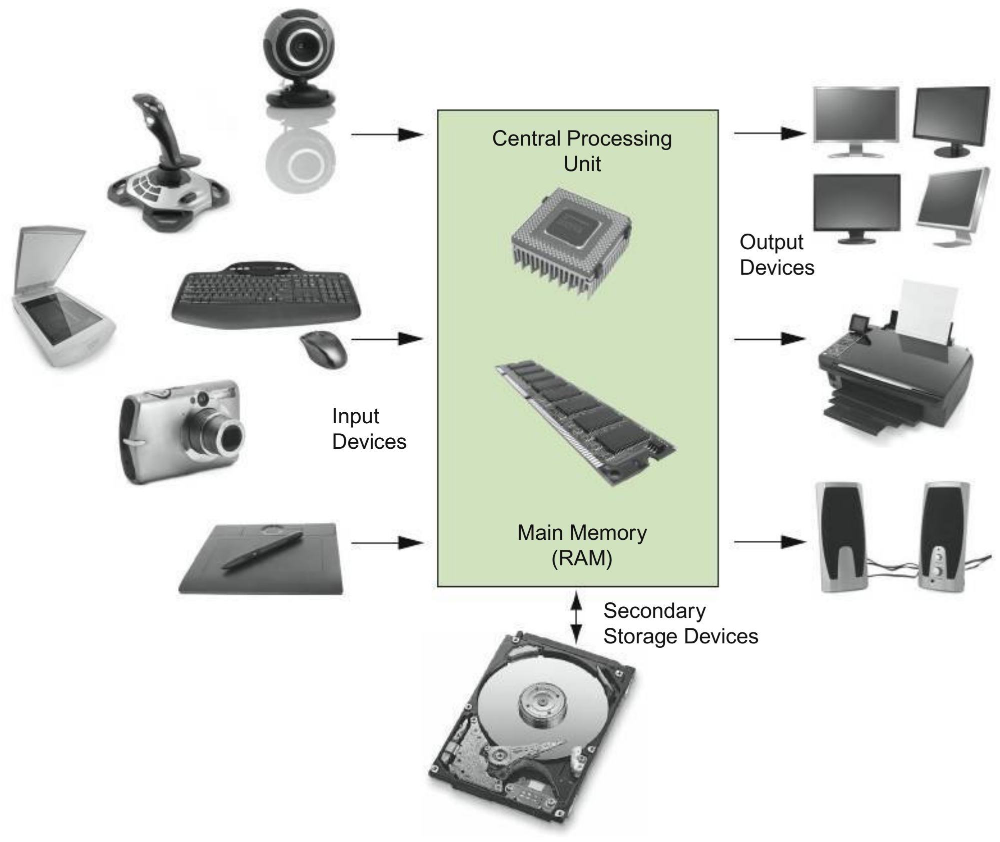
Hardware

The term **hardware** refers to a computer's physical components. A computer, as we generally think of it, is not an individual device, but rather a system of devices. Like the instruments in a symphony orchestra, each device plays its own part. A typical computer system consists of the following major components:

1. The central processing unit (CPU)
2. Main memory
3. Secondary storage devices
4. Input devices
5. Output devices

The organization of a computer system is shown in Figure 1-1.

Figure 1-1 The organization of a computer system



1. The CPU

When a computer is performing the tasks that a program tells it to do, we say that the computer is running or executing the program. The **central processing unit**, or **CPU**, is the part of a computer that actually runs programs. The CPU is the most important component in a computer because without it, the computer could not run software.

A **program** is a set of instructions that a computer's CPU follows to perform a task. The program's instructions are stored in the computer's memory, and the CPU's job is to fetch those instructions, one by one, and carry out the operations that they command. In memory, the instructions are stored as a series of **binary numbers**. A binary number is a sequence of 1s and 0s, such as

```
11011011
```

This number has no apparent meaning to people, but to the computer it might be an instruction to multiply two numbers or read another value from memory.

2. Main Memory

You can think of **main memory** as the computer's work area. This is where the computer stores a program while the program is running, as well as the data that the program is working with. For example, suppose you are using a word processing program to write an essay for one of your classes. While you do this, both the word processing program and the essay are stored in main memory.

Main memory is commonly known as **random-access memory**, or **RAM**. It is called this because the CPU is able to quickly access data stored at any random location in RAM. RAM is usually a volatile type of memory that is used only for temporary storage while a program is running. When the computer is turned off, the contents of RAM are erased. Inside your computer, RAM is stored in microchips.

3. Secondary Storage

Secondary storage is a type of memory that can hold data for long periods of time—even when there is no power to the computer. Frequently used programs are stored in secondary memory and loaded into main memory as needed. Important data, such as word processing documents, payroll data, and inventory figures, is saved to secondary storage as well.

The most common type of secondary storage device is the **disk drive**. A traditional disk drive stores data by magnetically encoding it onto a spinning circular disk. **Solid-state drives**, which store data in solid-state memory, are increasingly becoming popular. A solid-state drive has no moving parts, and operates faster than a traditional disk drive.

Most computers have some sort of secondary storage device, either a traditional disk drive or a solid-state drive, mounted inside their case. External storage devices are also available, which connect to one of the computer's communication ports, or plug into a memory slot. External storage devices can be used to create backup copies of important data or to move data to another computer. For example, **USB (Universal Serial Bus) drives** and **SD (Secure Digital) memory cards** are small devices that appear to the system as disk drives. They are inexpensive, reliable, and small enough to be carried in your pocket.

Optical devices such as the CD (compact disc) and the DVD (digital versatile disc) are also popular for data storage. Data is not recorded magnetically on an optical disc, but is encoded as a series of pits on the disc surface. CD and DVD drives use a laser to detect the pits and thus read the encoded data. Optical discs hold large amounts of data, and because recordable CD and DVD drives are now commonplace, they are good mediums for creating backup copies of data.

4. Input Devices

Input is any data the computer collects from the world outside of the computer. The device that collects the data and sends it to the computer is called an **input device**. Common input devices are the keyboard, mouse, touchscreen, scanner, and digital camera. Disk drives and optical drives can also be considered input devices because programs and data are retrieved from them and loaded into the computer's memory.

5. Output Devices

Output is any data the computer sends to the world outside of the computer. It might be a sales report, a list of names, a graphic image, or a sound. The data is sent to an **output device**, which formats and presents it. Common output devices are screens and printers. Storage devices can also be considered output devices because the CPU sends data to them in order to be saved.

Software

Software refers to the programs that run on a computer. There are two general categories of software: operating systems and application software. An **operating system** or **OS** is a set of programs that manages the computer's hardware devices and controls their processes. Windows, Mac OS, Android, iOS, and Linux are popular operating systems.

Application software refers to programs that make the computer useful to the user. These programs, which are generally called applications, solve specific problems or perform general operations that satisfy the needs of the user. Word processing, spreadsheet, and database packages are all examples of application software. As you work through this book, you will develop application software using Visual Basic.

Checkpoint

1.1 List the five major hardware components of a computer system.

1.2 What is main memory? What is its purpose?

1.3 Explain why computers have both main memory and secondary storage.

1.4 What are the two general categories of software?

1.2 Programs and Programming Languages

CONCEPT: A program is a set of instructions a computer follows in order to perform a task. A programming language is a special language used to write computer programs.

What Is a Program?

Computers are designed to follow instructions. A computer program is a set of instructions that enables the computer to solve a problem or perform a task. For example, suppose we want the computer to calculate someone's gross pay—a *Wage Calculator* application. Figure 1-2 shows a list of things the computer should do.

Collectively, the instructions in Figure 1-2 are called an **algorithm**. An algorithm is a set of well-defined steps for performing a task or solving a problem. Notice these steps are

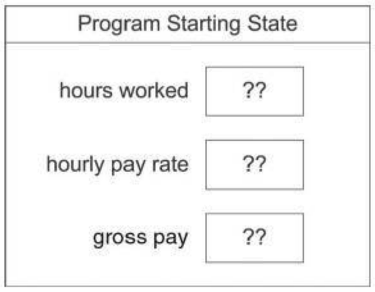
Figure 1-2 Program steps—*Wage Calculator* application

1. Display a message on the screen: *How many hours did you work?*
2. Allow the user to enter the number of hours worked.
3. Once the user enters a number, store it in memory.
4. Display a message on the screen: *How much do you get paid per hour?*
5. Allow the user to enter an hourly pay rate.
6. Once the user enters a number, store it in memory.
7. Once both the number of hours worked and the hourly pay rate are entered, multiply the two numbers and store the result in memory as the gross pay.
8. Display a message on the screen that shows the gross pay. The message must include the result of the calculation performed in Step 7.

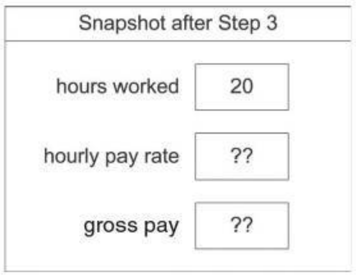
sequentially ordered. Step 1 should be performed before Step 2, and so on. It is important that these instructions are performed in their proper sequence.

States and Transitions

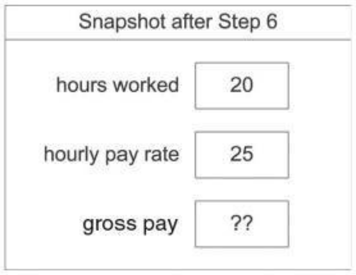
It is helpful to think of a running computer program as a combination of states and transitions. Each state is represented by a snapshot (like a picture) of the computer's memory. Using the *Wage Calculator* application example from Figure 1-2, the following is a memory snapshot taken when the program starts:



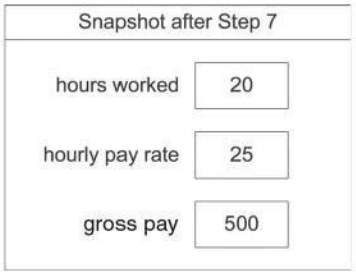
In Step 3, the number of hours worked by the user is stored in memory. Suppose the user enters the value 20. A new program state is created:



In Step 6, the hourly pay rate entered by the user is stored in memory. Suppose the user enters the value 25. The following memory snapshot shows the new program state:



In Step 7, the application calculates the amount of money earned, saving it in memory. The following memory snapshot shows the new program state:



The memory snapshot produced by Step 7 represents the final program state.

Programming Languages

In order for a computer to perform instructions such as the wage calculator algorithm, the steps must be converted to a format the computer can process. As mentioned earlier, a program is stored in memory as a series of binary numbers. These numbers are known as **machine language instructions**. The CPU processes only instructions written in machine language. Our *Wage Calculator* application might look like the following at the moment when it is executed by the computer:

```
1010110111010100011110000110111010001111000111001101010111 0 etc.
```

The CPU interprets these binary or machine language numbers as commands. As you might imagine, the process of encoding an algorithm in machine language is tedious and difficult. **Programming languages**, which use words instead of numbers, were invented to ease this task. Programmers can write their applications in programming language statements, and then use special software called a **compiler** to convert the program into machine language. Names of some popular recent programming languages are shown in Table 1-1. This list is only a small sample—there are thousands of programming languages.

Table 1-1 Popular programming languages

Language	Description
Visual Basic, C#	Popular programming languages for building Windows and Web applications.
C, C++	Powerful advanced programming languages that emphasize flexibility and fast running times. C++ is also object-oriented.
Java	Flexible and powerful programming language that runs on many different computer systems. Often used to teach object-oriented programming.
Python	Simple, yet powerful programming language used for graphics and small applications.
PHP	Programming language used for creating interactive Web sites.
JavaScript	Scripting language used in Web applications that provides rich user interfaces for Web browsers.

What Is a Program Made Of?

All programming languages, including Visual Basic, have certain elements in common. Let's look at the major programming language elements that you will work with when writing a program.

Keywords (Reserved Words)

Each high-level language has its own set of words that the programmer must learn in order to use the language. The words that make up a high-level programming language are known as **keywords** or **reserved words**. Each keyword has a predefined meaning and cannot be used for any other purpose. As you work through this book you will learn many of the Visual Basic keywords and how to use them in a program.

Operators

In addition to keywords, programming languages have **operators** that perform various operations on data. For example, all programming languages have math operators that perform arithmetic. In Visual Basic, as well as most other languages, the + sign is an operator that adds two numbers. The following would add 12 and 75:

```
12 + 75
```

Variables

Programs use variables to store data in memory. A **variable** is a storage location in memory that is represented by a name. When a value is stored in a variable, it is stored in the computer's memory.

Programmers make up the names for all the variables that they use in a program. You will learn specific rules and guidelines for naming variables in Chapter 3, but for now just remember that a variable's name is a single word that indicates what the variable is used for. For example, a program that calculates the sales tax on a purchase might use a variable named `tax` to hold that value in memory. And a program that calculates the distance from Earth to a star might use a variable named `distance` to hold that value in memory. When a program stores a value in a variable, the value is actually stored in memory at the location represented by the variable.

Syntax

In addition to keywords and operators, each language also has its own **syntax**, which is a set of rules that must be strictly followed when writing a program. The syntax rules dictate how keywords, operators, and various punctuation characters must be used in a program. When you are learning a programming language, you must learn the syntax rules for that particular language.

NOTE: Human languages also have syntax rules. Do you remember when you took your first English class, and you learned all those rules about infinitives, indirect objects, clauses, and so forth? You were learning the syntax of the English language.

Although people commonly violate the syntax rules of their native language when speaking and writing, other people usually understand what they mean. Unfortunately, program compilers do not have this ability. If even a single syntax error appears in a program, the program cannot be compiled or executed.

Statements

The individual instructions that you write in a program are called **statements**. A programming statement can consist of keywords, operators, punctuation, and other allowable programming elements, arranged in the proper sequence to perform an operation. The statements that are written in a program are commonly called **source code**, or simply **code**.

Procedures

A **procedure** is a set of programming statements that exist within a program for the purpose of performing a specific task. The program executes the procedure when the task needs to be performed.

Comments (Remarks)

Not everything a programmer writes in a program is meant to be executed by the computer. Some parts of a program are **comments**, or **remarks,** that help the human reader of a program understand the purposes of program statements. In Visual Basic, any statement that begins with an apostrophe (') is considered a comment. When the Visual Basic compiler sees a statement that begins with an apostrophe, it recognizes it as a comment and it skips over it.

You should always add descriptive comments to your code. The extra time it takes is well spent. Sometimes you (the programmer) will have to reread and understand your own code. Comments are a great way to remind you of what you were thinking when you created the program. In addition, you may have to modify or maintain code written by another programmer and you will appreciate the time spent to write comments!

Graphical User Interfaces

When a computer program is needed to perform a task, a programmer is the person who develops the algorithm, and writes the programming statements that perform the algorithm's steps. Once the program is complete, it is made available to those who need to use it. The people who use the program are known as **users**.

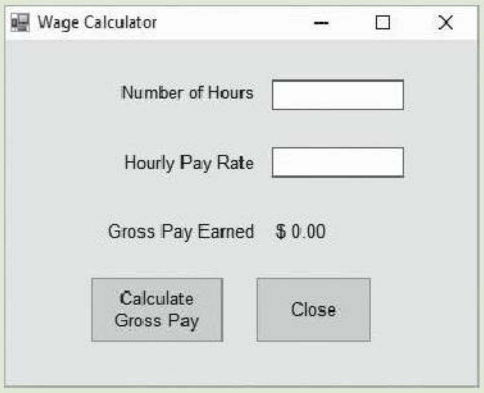
Although a programmer works directly with a program's statements, users are typically not concerned with the program's inner workings. Users want to make sure they know how to operate the program when it is running, and that the program works as it should. The part of a program that users interact with is known as the **user interface**. On modern operating systems such as Windows, most of the programs that people use have a **graphical user interface**, or **GUI** (pronounced *gooey*). A graphical user interface typically consists of one or more windows that appear on the computer screen. A **window** is a rectangular area that contains other visual elements such as text, buttons that can be clicked with the mouse, boxes that accept keyboard input, and so forth. Let's look at an example. Follow the steps in Tutorial 1-1 to run a program that you can download from the book's companion Website, at www.pearsonhighered.com/gaddisvb.

Tutorial 1-1:
Running the *Wage Calculator* application

Step 1: Make sure you have downloaded the student sample programs from the textbook's companion Website, at www.pearsonhighered.com/gaddisvb. If you are working in your school's computer lab, your instructor will tell you where the files are located.

Step 2: Go to the folder containing the student sample programs for Chapter 1. Double-click the file *Wage Calculator.exe* (the *.exe* filename extension may not be visible). The program's window should display as shown in Figure 1-3. (Figure 1-3 shows the application running on a Windows 10 system. If you are using another version of Windows, the screen might appear differently.) Leave the program running as you continue to read. We will perform operations with the program in Tutorial 1-2.

Figure 1-3 A graphical user interface



The program you executed in Tutorial 1-1 calculates an employee's gross pay. Notice that inside the program's window (shown in Figure 1-3) there are boxes for entering the number of hours worked and the hourly pay rate. There is also a button that calculates the gross pay when it is clicked with the mouse, and a button that closes the program (stops its execution). All of these elements are part of the program's GUI, and anyone operating the program will interact with these elements.

Objects and Controls

As a student studying Visual Basic, you will frequently encounter two terms: object and control. An **object** is an item in a program that contains data and has the ability to perform operations. The data an object contains is referred to as **properties**, or **attributes**. The operations an object can perform are called **methods**. (Recall that earlier we mentioned that a procedure is a set of programming statements that exist within a program for the purpose of performing a specific task. A method is a special type of procedure that belongs to an object.)

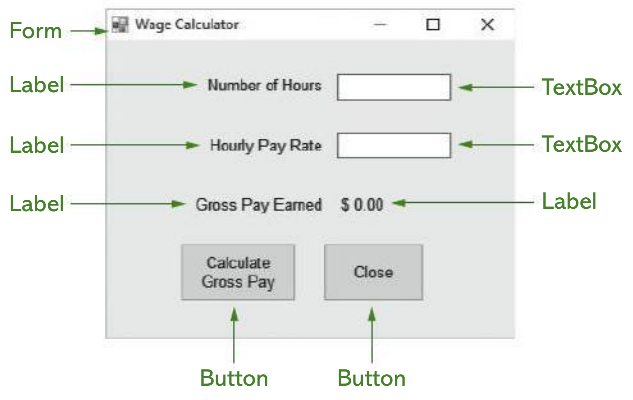
In the beginning of your studies you will learn how to use many different objects that are provided by Visual Basic to perform various operations in your programs. In Chapter 12, you will learn to define your own objects.

VideoNote

Forms, Controls, and Properties

A **control** is a specific type of object that usually appears in a program's graphical user interface. For example, each element appearing in the user interface in Figure 1-3 is a control. The window that contains the other elements is known as a **Form** control. The small boxes that accept keyboard input are known as **TextBox** controls. The areas that simply display text are known as **Label** controls. The buttons that perform operations when clicked with the mouse are known as **Button** controls. Figure 1-4 points out each of these controls in the user interface.

Figure 1-4 Types of controls



NOTE: Visual Basic is an **object-oriented programming** (**OOP**) language. A typical VB application uses numerous objects (such as GUI controls) that work together.

Properties

A GUI control's visual appearance is determined by the control's properties. A **property** is a piece of data that determines some characteristic of the control. For example, many controls have a **Text property** that determines the text that is displayed by the control. If you look at Figure 1-4, near the top of the form you see a Label control that displays the text *Number of Hours*. That Label control's Text property is set to the value *Number of*