starting out with >>>

PROGRAMMING LOGIC AND DESIGN

3RD EDITION

TONY GADDIS

Third Edition

Starting Out with

Programming Logic & Design

This page intentionally left blank

Third Edition

Starting Out with

Programming Logic & Design

Tony Gaddis

Haywood Community College

PEARSON

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

Editorial Director: Marcia Horton
Editor in Chief: Michael Hirsch
Acquisitions Editor: Matt Goldstein
Editorial Assistant: Chelsea Kharakozova
Director of Marketing: Patrice Jones
Marketing Manager: Yez Alayan
Marketing Coordinator: Kathryn Ferranti
Marketing Assistant: Emma Snider
Director of Production: Vince O'Brien
Managing Editor: Jeff Holcomb
Production Editor: Pat Brown
Manufacturing Buyer: Pat Brown
Art Director: Anthony Gemmellaro
Cover Designer: Joyce Cosentino Wells
Cover Art: © iStockphoto
Media Project Manager: John Cassar
Full-Service Project Management: Jogender Taneja/Aptara®, Inc.
Composition: Aptara®, Inc.
Printer/Bindery: Edwards Brothers
Cover Printer: Lehigh-Phoenix Color/Hagerstown

Library of Congress Cataloging-in-Publication Data

Gaddis, Tony.
Starting out with programming logic and design/Tony Gaddis. — 3rd ed.
p. cm.
Includes index.
ISBN-13: 978-0-13-280545-2
ISBN-10: 0-13-280545-6
1. Computer programming. I. Title. II. Title: Starting out with programming logic and design.
QA76.6.G315 2013
005.1—dc23 2011044250

10 9 8 7 6 5 4 3 2 1

PEARSON

ISBN 10: 0-13-280545-6
ISBN 13: 978-0-13-280545-2

Brief Contents

This page intentionally left blank

Contents

This page intentionally left blank

Preface

Welcome to *Starting Out with Programming Logic and Design*, Third Edition. This book uses a language-independent approach to teach programming concepts and problem-solving skills, without assuming any previous programming experience. By using easy-to-understand pseudocode, flowcharts, and other tools, the student learns how to design the logic of programs without the complication of language syntax.

Fundamental topics such as data types, variables, input, output, control structures, modules, functions, arrays, and files are covered as well as object-oriented concepts, GUI development, and event-driven programming. As with all the books in the *Starting Out With . . .* series, this text is written in clear, easy-to-understand language that students find friendly and inviting.

Each chapter presents a multitude of program design examples. Short examples that highlight specific programming topics are provided, as well as more involved examples that focus on problem solving. Each chapter includes at least one *In the Spotlight section* that provides step-by-step analysis of a specific problem and demonstrates a solution to that problem.

This book is ideal for a programming logic course that is taught as a precursor to a language-specific introductory programming course, or for the first part of an introductory programming course in which a specific language is taught.

Changes in the Third Edition

This book's pedagogy, organization, and clear writing style remain the same as in the previous edition. Many improvements have been made, which are summarized here:

- **Detailed guidance for students designing their first program**

  A new section titled *Designing Your First Program* has been added to Chapter 2. This section takes the student through the process of analyzing a problem and determining its requirements. The student sees an example of how a program's input, processing, and output can be determined, as a prelude to writing pseudocode and drawing flowcharts.

  Also, a new *In the Spotlight* section has been added to Chapter 2 to show the student how to examine the steps that are taken to manually perform a calculation (determining cell phone overage fees), and then convert those steps to a computer algorithm.

- **New Debugging Exercises**

  A new set of *Debugging Exercises* have been added to most of the chapters. The student examines a set of pseudocode algorithms and identifies logical errors.

- **Greater consistency between flowcharts and pseudocode**

  Throughout the book, many of the flowcharts have been revised so they appear more consistent with the pseudocode.

- **Expanded coverage of nested repetition structures**

  In Chapter 5 the section on nested loops has been expanded with an additional example.

- **Additional VideoNotes for repetition structures**

  New VideoNotes have been added for the `Do-While` and `For` loops in Chapter 5.

- **File specification documentation and print spacing charts**

  File specification documentation and print spacing charts are now discussed in Chapter 10.

- **New pseudocode quick reference guide**

  A quick reference guide to the pseudocode used in the book has been added as Appendix C.

- **New Programming Language Companions**

  New language companions have been added for Python 3 and C++. All of the book's language companions are available on the book's resource site at `www.pearsonhighered.com/gaddis`.

Brief Overview of Each Chapter

Chapter 1: Introduction to Computers and Programming

This chapter begins by giving a concise and easy-to-understand explanation of how computers work, how data is stored and manipulated, and why we write programs in high-level languages.

Chapter 2: Input, Processing, and Output

This chapter introduces the program development cycle, data types, variables, and sequence structures. The student learns to use pseudocode and flowcharts to design simple programs that read input, perform mathematical operations, and produce screen output.

Chapter 3: Modules

This chapter demonstrates the benefits of modularizing programs and using the top-down design approach. The student learns to define and call modules, pass arguments to modules, and use local variables. Hierarchy charts are introduced as a design tool.

Chapter 4: Decision Structures and Boolean Logic

In this chapter students explore relational operators and Boolean expressions and are shown how to control the flow of a program with decision structures. The `If-Then,`

`If-Then-Else`, and `If-Then-Else If` statements are covered. Nested decision structures, logical operators, and the case structure are also discussed.

Chapter 5: Repetition Structures

This chapter shows the student how to use loops to create repetition structures. The `While, Do-While, Do-Until,` and `For` loops are presented. Counters, accumulators, running totals, and sentinels are also discussed.

Chapter 6: Functions

This chapter begins by discussing common library functions, such as those for generating random numbers. After learning how to call library functions and how to use values returned by functions, the student learns how to define and call his or her own functions.

Chapter 7: Input Validation

This chapter discusses the importance of validating user input. The student learns to write input validation loops that serve as error traps. Defensive programming and the importance of anticipating obvious as well as unobvious errors is discussed.

Chapter 8: Arrays

In this chapter the student learns to create and work with one- and two-dimensional arrays. Many examples of array processing are provided including examples illustrating how to find the sum, average, and highest and lowest values in an array, and how to sum the rows, columns, and all elements of a two-dimensional array. Programming techniques using parallel arrays are also demonstrated.

Chapter 9: Sorting and Searching Arrays

In this chapter the student learns the basics of sorting arrays and searching for data stored in them. The chapter covers the bubble sort, selection sort, insertion sort, and binary search algorithms.

Chapter 10: Files

This chapter introduces sequential file input and output. The student learns to read and write large sets of data, store data as fields and records, and design programs that work with both files and arrays. The chapter concludes by discussing control break processing.

Chapter 11: Menu-Driven Programs

In this chapter the student learns to design programs that display menus and execute tasks according to the user's menu selection. The importance of modularizing a menu-driven program is also discussed.

Chapter 12: Text Processing

This chapter discusses text processing at a detailed level. Algorithms that step through the individual characters in a string are discussed, and several common library functions for character and text processing are introduced.

Chapter 13: Recursion

This chapter discusses recursion and its use in problem solving. A visual trace of recursive calls is provided, and recursive applications are discussed. Recursive algorithms for many tasks are presented, such as finding factorials, finding a greatest common denominator (GCD), summing a range of values in an array, and performing a binary search. The classic Towers of Hanoi example is also presented.

Chapter 14: Object-Oriented Programming

This chapter compares procedural and object-oriented programming practices. It covers the fundamental concepts of classes and objects. Fields, methods, access specification, constructors, accessors, and mutators are discussed. The student learns how to model classes with UML and how to find the classes in a particular problem.

Chapter 15: GUI Applications and Event-Driven Programming

This chapter discusses the basic aspects of designing a GUI application. Building graphical user interfaces with visual design tools (such as Visual Studio® or NetBeans™) is discussed. The student learns how events work in a GUI application and how to write event handlers.

Appendix A: ASCII/Unicode Characters

This appendix lists the ASCII character set, which is the same as the first 127 Unicode character codes.

Appendix B: Flowchart Symbols

This appendix shows the flowchart symbols that are used in this book.

Appendix C: Pseudocode Reference

This appendix provides a quick reference for the pseudocode language that is used in the book.

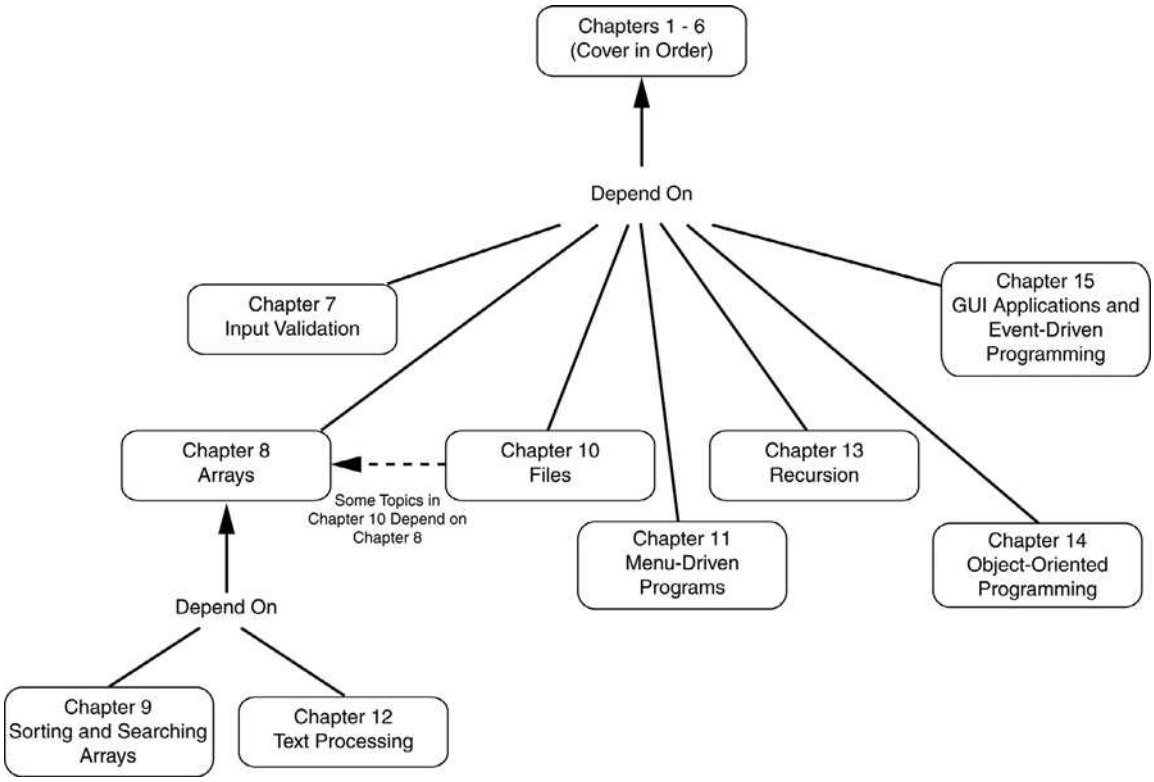
Appendix D: Answers to Checkpoint Questions

This appendix provides answers to the Checkpoint questions that appear throughout the text, and can be downloaded from the CD that accompanies this book or from the book's online resource page at `www.pearsonhighered.com/gaddis`.

Organization of the Text

The text teaches programming logic and design in a step-by-step manner. Each chapter covers a major set of topics and builds knowledge as students progress through the book. Although the chapters can be easily taught in their existing sequence, there is some flexibility. Figure P-1 shows chapter dependencies. Each box represents a chapter or a group of chapters. A chapter to which an arrow points must be covered before the chapter from which the arrow originates. The dotted line indicates that only a portion of Chapter 10 depends on information presented in Chapter 8.

Figure P-1 Chapter dependencies



Features of the Text

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

Example Programs. Each chapter has an abundant number of complete and partial example programs, each designed to highlight the current topic. Pseudocode, flowcharts, and other design tools are used in the example programs.

In the Spotlight. Each chapter has one or more *In the Spotlight* case studies that provide detailed, step-by-step analysis of problems, and show the student how to solve them.

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

NOTE: Notes appear at several places throughout the text. They are short explanations of interesting or often misunderstood points relevant to the topic at hand.

TIP: Tips advise the student on the best techniques for approaching different programming or animation problems.

WARNING! Warnings caution students about programming techniques or practices that can lead to malfunctioning programs or lost data.

Programming Language Companions. Many of the pseudocode programs shown in this book have also been written in Java, Python, and Visual Basic. These programs appear in the programming language companions that are available at `www.pearsonhighered.com/gaddis`. Icons appear next to each pseudocode program that also appears in the language companions.

Checkpoints. Checkpoints are questions placed at intervals throughout each chapter. They are designed to query the student's knowledge quickly after learning a new topic.

Review Questions. Each chapter presents a thorough and diverse set of Review Questions and exercises. They include Multiple Choice, True/False, Short Answer, and Algorithm Workbench.

Debugging Exercises. Most chapters provide a set of debugging exercises in which the student examines a set of pseudocode algorithms and identifies logical errors.

Programming Exercises. Each chapter offers a pool of Programming Exercises designed to solidify the student's knowledge of the topics currently being studied.

Supplements

Student Online Resources

Many student resources are available for this book from the publisher. The following items are available on the Gaddis Series resource page at `www.pearsonhighered.com/gaddis`:

- **Access to the book's companion VideoNotes**

  An extensive series of online VideoNotes have been developed to accompany this text. Throughout the book, VideoNote icons alert the student to videos covering specific topics. Additionally, one programming exercise at the end of each chapter has an accompanying VideoNote explaining how to develop the problem's solution.

- **Access to the Language Companions for Python, Java, Visual Basic, and C++**

  Programming language companions specifically designed to accompany the Third Edition of this textbook are available for download. The companions introduce the Java™, Python®, Visual Basic®, and C++ programming languages, and correspond on a chapter-by-chapter basis with the textbook. Many of the pseudocode programs that appear in the textbook also appear in the companions, implemented in a specific programming language.

- **A link to download the RAPTOR flowcharting environment**

  RAPTOR is a flowchart-based programming environment developed by the US Air Force Academy Department of Computer Science.

Instructor Resources

The following supplements are available to qualified instructors only:

- Answers to all of the Review Questions
- Solutions for the Programming Exercises
- PowerPoint® presentation slides for each chapter
- Test bank

Visit the Pearson Instructor Resource Center (`http://www.pearsonhighered.com/irc`) or send an email to `computing@aw.com` for information on how to access them.

This page intentionally left blank

Acknowledgments

There have been many helping hands in the development and publication of this text. I would like to thank the following faculty reviewers:

Reviewers of the Second Edition

Cherie Aukland
Thomas Nelson Community College

Steve Browning
Freed Hardeman University

Stephen Robert Cheskiewicz
Keystone College and Wilkes University

Ronald J. Harkins
Miami University, OH

Robert S. Overall, III
Nashville State Community College

John Thacher
Gwinnett Technical College

Scott VanSelow
Edison State College

Reviewers of the First Edition

Reni Abraham
Houston Community College

John P. Buerck
Saint Louis University

Jill Canine
Ivy Tech Community College of Indiana

Steven D. Carver
Ivy Tech Community College

Katie Danko
Grand Rapids Community College

Coronicca Oliver
Coastal Georgia Community College

Dale T. Pickett
Baker College of Clinton Township

Tonya Pierce
Ivy Tech Community College

Larry Strain
Ivy Tech Community College–Bloomington

Donald Stroup
Ivy Tech Community College

Jim Turney
Austin Community College

I also want to thank everyone at Pearson for making the *Starting Out With . . .* series so successful. I have worked so closely with the team at Pearson Addison-Wesley that I consider them among my closest friends. I am extremely fortunate to have Michael Hirsch and Matt Goldstein as my editors, and Chelsea Kharakozova as Editorial Assistant. They have guided me through the process of revising this, and many other books. I am also fortunate to have Yez Alayan as Marketing Manager, and Kathryn Ferranti as Marketing Coordinator. Their hard work is truly inspiring, and they do a great job getting my books out to the academic community. The production team of Jeff Holcomb and Pat Brown worked tirelessly to make this book a reality. Thanks to you all!

About the Author

Tony Gaddis is the principal author of the *Starting Out With . . .* series of textbooks. Tony has twenty years of experience teaching computer science courses, primarily at Haywood Community College. He is a highly acclaimed instructor who was previously selected as the North Carolina Community College "Teacher of the Year" and has received the Teaching Excellence award from the National Institute for Staff and Organizational Development. The *Starting Out With . . .* series includes introductory books covering Programming Logic and Design, C++, Java, Microsoft® Visual Basic, C#®, Python, and Alice, all published by Pearson.

This page intentionally left blank

Third Edition

Starting Out with

Programming Logic & Design

This page intentionally left blank

CHAPTER 1

Introduction to Computers and Programming

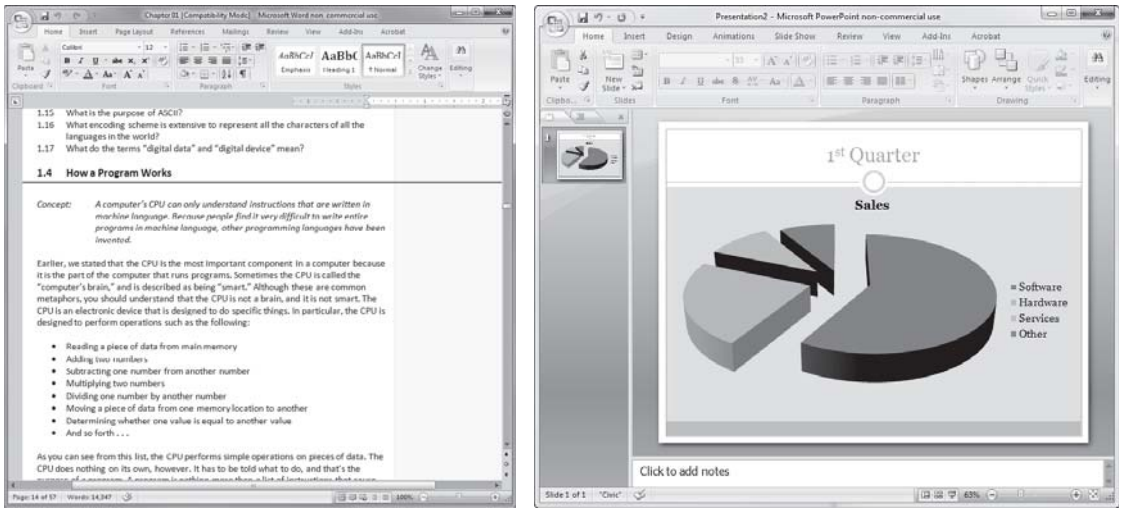
TOPICS

1.1 Introduction

Think about some of the different ways that people use computers. In school, students use computers for tasks such as writing papers, searching for articles, sending email, and participating in online classes. At work, people use computers to analyze data, make presentations, conduct business transactions, communicate with customers and coworkers, control machines in manufacturing facilities, and many other things. At home, people use computers for tasks such as paying bills, shopping online, communicating with friends and family, and playing computer games. And don't forget that cell phones, iPods®, BlackBerries®, car navigation systems, and many other devices are computers too. The uses of computers are almost limitless in our everyday lives.

Computers can do such a wide variety of things because they can be programmed. This means that computers are not designed to do just one job, but to do any job that their programs tell them to do. A *program* is a set of instructions that a computer follows to perform a task. For example, Figure 1-1 shows screens from two commonly used programs: Microsoft Word and PowerPoint.

Figure 1-1 Commonly used programs



Programs are commonly referred to as *software*. Software is essential to a computer because without software, a computer can do nothing. All of the software that we use to make our computers useful is created by individuals known as programmers or software developers. A *programmer,* or *software developer,* is a person with the training and skills necessary to design, create, and test computer programs. Computer programming is an exciting and rewarding career. Today, you will find programmers working in business, medicine, government, law enforcement, agriculture, academics, entertainment, and almost every other field.

This book introduces you to the fundamental concepts of computer programming. Before we begin exploring those concepts, you need to understand a few basic things about computers and how they work. This chapter will build a solid foundation of knowledge that you will continually rely on as you study computer science. First, we will discuss the physical components that computers are commonly made of. Next, we will look at how computers store data and execute programs. Finally, we will discuss the major types of software that computers use.

1.2 Hardware

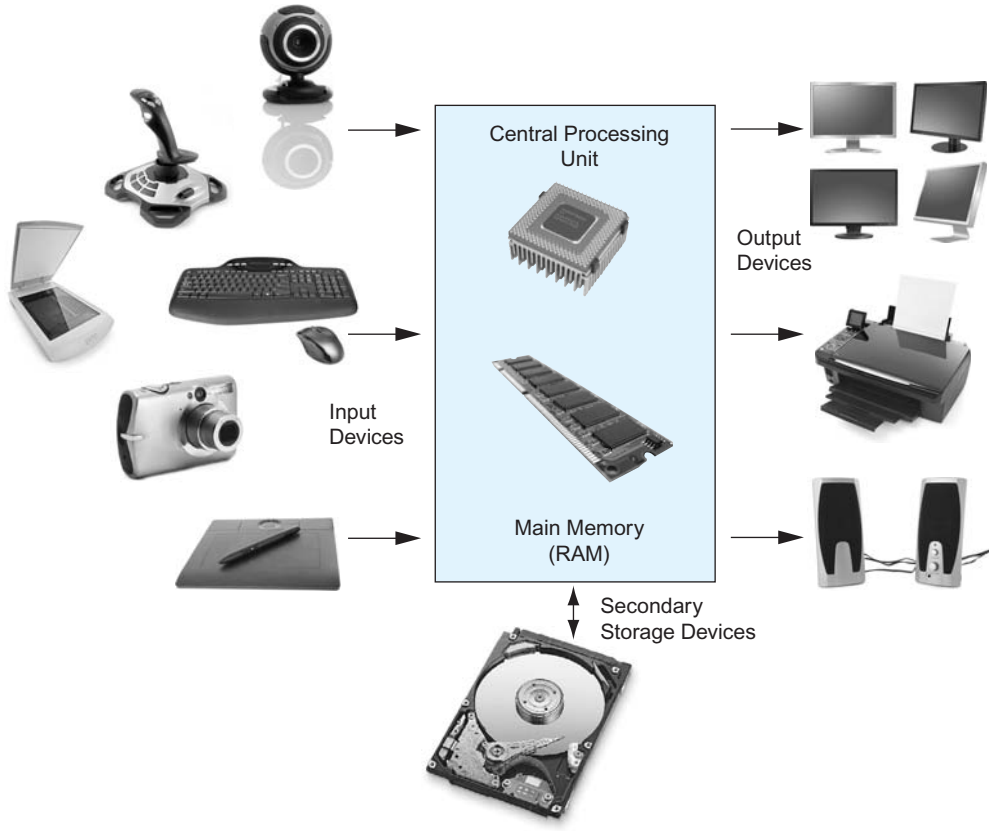
CONCEPT: **The physical devices that a computer is made of are referred to as the computer's hardware. Most computer systems are made of similar hardware devices.**

The term *hardware* refers to all of the physical devices, or *components,* that a computer is made of. A computer is not one single device, but a system of devices that all work together. Like the different instruments in a symphony orchestra, each device in a computer plays its own part.

If you have ever shopped for a computer, you've probably seen sales literature listing components such as microprocessors, memory, disk drives, video displays, graphics cards, and so on. Unless you already know a lot about computers, or at least have a friend who does, understanding what these different components do can be confusing. As shown in Figure 1-2, a typical computer system consists of the following major components:

- The central processing unit (CPU)
- Main memory
- Secondary storage devices
- Input devices
- Output devices

Figure 1-2 Typical components of a computer system (all photos © Shutterstock)



Let's take a closer look at each of these components.

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.

In the earliest computers, CPUs were huge devices made of electrical and mechanical components such as vacuum tubes and switches. Figure 1-3 shows such a device. The two women in the photo are working with the historic ENIAC computer. The *ENIAC,* considered by many to be the world's first programmable electronic computer, was built in 1945 to calculate artillery ballistic tables for the U.S. Army. This machine, which was primarily one big CPU, was 8 feet tall, 100 feet long, and weighed 30 tons.

Today, CPUs are small chips known as *microprocessors*. Figure 1-4 shows a photo of a lab technician holding a modern-day microprocessor. In addition to being much smaller than the old electro-mechanical CPUs in early computers, microprocessors are also much more powerful.

Figure 1-3 The ENIAC computer (photo courtesy of U.S. Army Historic Computer Images)

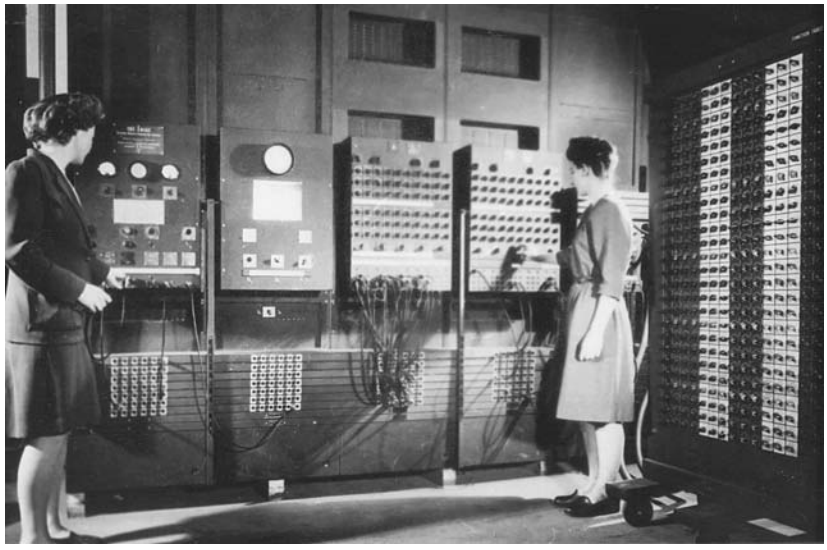


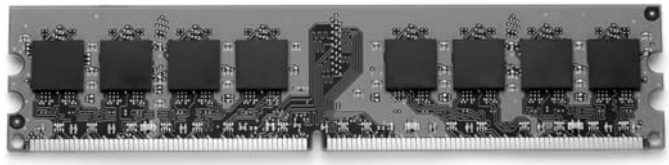
Figure 1-4 A lab technician holds a modern microprocessor (photo courtesy of Intel Corporation)

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 chips, similar to the ones shown in Figure 1-5.

Figure 1-5 Memory chips (photo © Garsya/Shutterstock)



Secondary Storage Devices

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. Programs are normally stored in secondary memory and loaded into main memory as needed. Important data, such as word processing documents, payroll data, and inventory records, is saved to secondary storage as well.

The most common type of secondary storage device is the disk drive. A *disk drive* stores data by magnetically encoding it onto a circular disk. Most computers have a disk drive mounted inside their case. External disk drives, which connect to one of the computer's communication ports, are also available. External disk drives can be used to create backup copies of important data or to move data to another computer.

In addition to external disk drives, many types of devices have been created for copying data, and for moving it to other computers. For many years floppy disk drives were popular. A *floppy disk drive* records data onto a small floppy disk, which can be removed from the drive. Floppy disks have many disadvantages, however. They hold only a small amount of data, are slow to access data, and are sometimes unreliable. The use of floppy disk drives has declined dramatically in recent years, in favor of

superior devices such as USB drives. *USB drives* are small devices that plug into the computer's USB (universal serial bus) port, and appear to the system as a disk drive. These drives do not actually contain a disk, however. They store data in a special type of memory known as *flash memory*. USB drives, which are also known as *memory sticks* and *flash drives,* 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.

Input Devices

Input is any data the computer collects from people and from other devices. The component that collects the data and sends it to the computer is called an *input device*. Common input devices are the keyboard, mouse, scanner, microphone, 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.

Output Devices

Output is any data the computer produces for people or for other devices. It might be a sales report, a list of names, or a graphic image. The data is sent to an *output device,* which formats and presents it. Common output devices are video displays and printers. Disk drives and CD recorders can also be considered output devices because the system sends data to them in order to be saved.

Checkpoint

1.1 What is a program?

1.2 What is hardware?

1.3 List the five major components of a computer system.

1.4 What part of the computer actually runs programs?

1.5 What part of the computer serves as a work area to store a program and its data while the program is running?

1.6 What part of the computer holds data for long periods of time, even when there is no power to the computer?

1.7 What part of the computer collects data from people and from other devices?

1.8 What part of the computer formats and presents data for people or other devices?

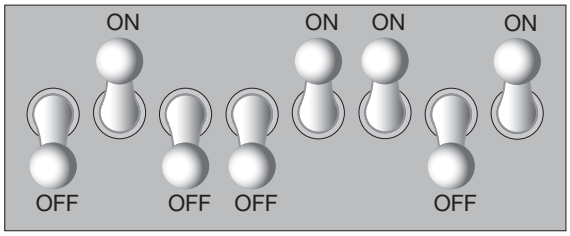
1.3 How Computers Store Data

CONCEPT: **All data that is stored in a computer is converted to sequences of 0s and 1s.**

A computer's memory is divided into tiny storage locations known as *bytes*. One byte is only enough memory to store a letter of the alphabet or a small number. In order to do anything meaningful, a computer has to have lots of bytes. Most computers today have millions, or even billions, of bytes of memory.

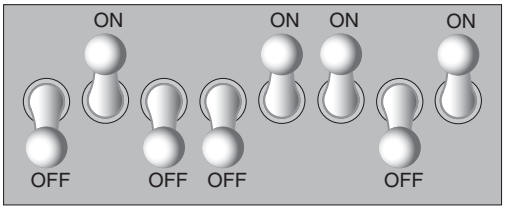
Each byte is divided into eight smaller storage locations known as bits. The term *bit* stands for *binary digit*. Computer scientists usually think of bits as tiny switches that can be either on or off. Bits aren't actual "switches," however, at least not in the conventional sense. In most computer systems, bits are tiny electrical components that can hold either a positive or a negative charge. Computer scientists think of a positive charge as a switch in the *on* position, and a negative charge as a switch in the *off* position. Figure 1-6 shows the way that a computer scientist might think of a byte of memory: as a collection of switches that are each flipped to either the on or off position.

Figure 1-6 Think of a byte as eight switches

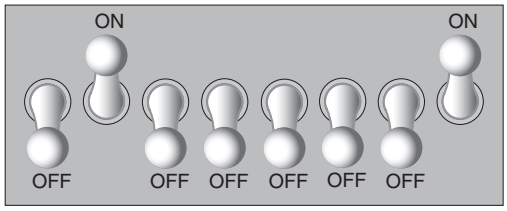


When a piece of data is stored in a byte, the computer sets the eight bits to an on/off pattern that represents the data. For example, the pattern shown on the left in Figure 1-7 shows how the number 77 would be stored in a byte, and the pattern on the right shows how the letter A would be stored in a byte. In a moment you will see how these patterns are determined.

Figure 1-7 Bit patterns for the number 77 and the letter A



The number 77 stored in a byte.



The letter A stored in a byte.

Storing Numbers

A bit can be used in a very limited way to represent numbers. Depending on whether the bit is turned on or off, it can represent one of two different values. In computer systems, a bit that is turned off represents the number 0 and a bit that is turned on represents the number 1. This corresponds perfectly to the *binary numbering system*. In the binary numbering system (or *binary,* as it is usually called) all numeric values are written as sequences of 0s and 1s. Here is an example of a number that is written in binary:

```
10011101
```

The position of each digit in a binary number has a value assigned to it. Starting with the rightmost digit and moving left, the position values are 2^0, 2^1, 2^2, 2^3, and so forth, as shown in Figure 1-8. Figure 1-9 shows the same diagram with the position values calculated. Starting with the rightmost digit and moving left, the position values are 1, 2, 4, 8, and so forth.

Figure 1-8 The values of binary digits as powers of 2

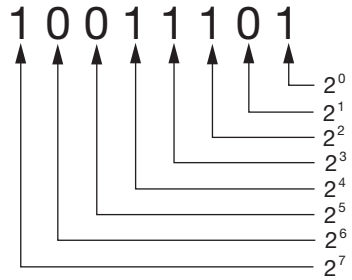
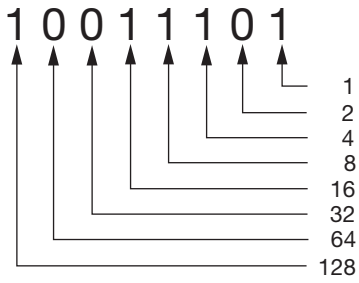


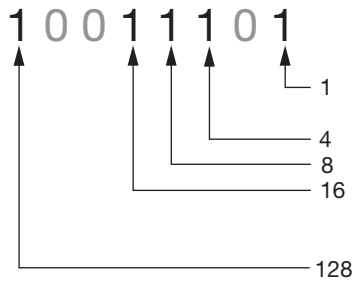
Figure 1-9 The values of binary digits



To determine the value of a binary number you simply add up the position values of all the 1s. For example, in the binary number 10011101, the position values of the 1s are 1, 4, 8, 16, and 128. This is shown in Figure 1-10. The sum of all of these position values is 157. So, the value of the binary number 10011101 is 157.

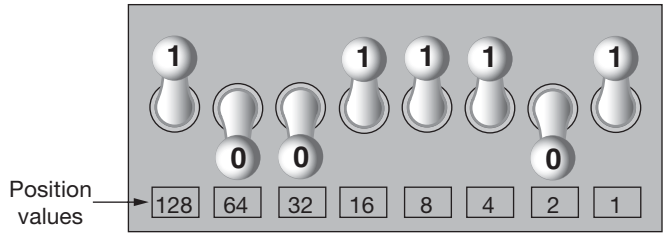
Figure 1-11 shows how you can picture the number 157 stored in a byte of memory. Each 1 is represented by a bit in the on position, and each 0 is represented by a bit in the off position.

Figure 1-10 Determining the value of 10011101



1 + 4 + 8 + 16 + 128 = **157**

Figure 1-11 The bit pattern for 157

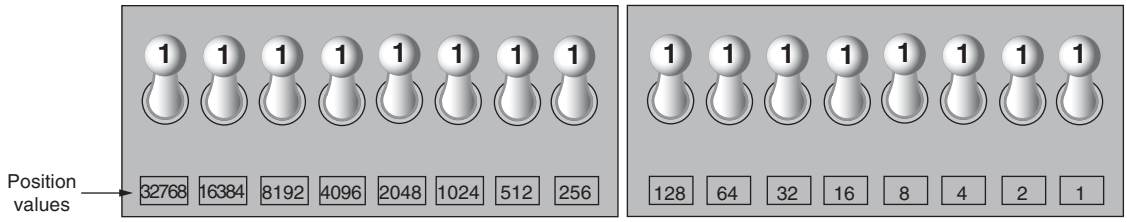


128 + 16 + 8 + 4 + 1 = **157**

When all of the bits in a byte are set to 0 (turned off), then the value of the byte is 0. When all of the bits in a byte are set to 1 (turned on), then the byte holds the largest value that can be stored in it. The largest value that can be stored in a byte is 1 + 2 + 4 + 8 + 16 + 32 + 64 + 128 = 255. This limit exists because there are only eight bits in a byte.

What if you need to store a number larger than 255? The answer is simple: use more than one byte. For example, suppose we put two bytes together. That gives us 16 bits. The position values of those 16 bits would be 2^0, 2^1, 2^2, 2^3, and so forth, up through 2^{15}. As shown in Figure 1-12, the maximum value that can be stored in two bytes is 65,535. If you need to store a number larger than this, then more bytes are necessary.

Figure 1-12 Two bytes used for a large number



32768 + 16384 + 8192 + 4096 + 2048 + 1024 + 512 + 256 + 128 + 64 + 32 + 16 + 8 + 4 + 2 + 1 = **65535**