Prelude to Programming

Concepts and Design

Sixth Edition

Stewart Venit | Elizabeth Drake

Prelude to Programming

Concepts and Design

Sixth Edition

Stewart Venit | Elizabeth Drake

PEARSON

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

Editorial Director: Marcia Horton
Acquisitions Editor: Matt Goldstein
Program Manager: Kayla Smith-Trabox
Director of Marketing: Christy Lesko
Marketing Manager: Yez Alayan
Marketing Coordinator: Kathryn Ferranti
Marketing Assistant: Jon Bryant
Senior Managing Editor: Scott Disanno
Senior Project Manager: Marilyn Lloyd
Operations Supervisor: Vincent Scelta
Operations Specialist: Linda Sager
Art Director, Cover: Jayne Conte
Text Designer: Gillian Hall
Cover Designer: Bruce Kenselaar
Manager, Visual Research: Karen Sanatar
Permissions Supervisor: Michael Joyce
Permission Administrator: Jenell Forschler
Cover Image: © Bombaert Patrick / Fotolia
Media Project Manager: Renata Butera
Full-Service Project Manager: Haseen Khan/ Laserwords, Pvt. Ltd
Full-Service Vendor: Laserwords, Pvt. Ltd
Printer/Binder: Courier Kendallville
Cover Printer: Lehigh-Phoenix Color/Hagerstown

Credits:

Cover: © Bombaert Patrick / Fotolia; Figure 0.1 U. S. Army Center of Military History; Figure 0.2 dule964/Fotolia; Figure 0.3 Shutterstock/Stu49; Figure 0.4a Jultud/Fotolia; Figure 0.4b Giuseppe Lancia/Fotolia; Figure 0.5 Fotosearch/ Publitek, Inc.; Figure 0.7 National Center for Computational Sciences; Figure 6a Chuck/Alamy; Figure 6b Marian Stanca/Alamy; Figure 11.01a Shutterstock; Figure 11.01b Shutterstock

Screenshots throughout the entire text: RAPTOR is provided free courtesy of the United States Air Force Academy, http://raptor.martincarlisle.com/

Library of Congress Cataloging-in-Publication Data will be available upon request

10 9 8 7 6 5 4 3 2 1

PEARSON

ISBN 10: 0-13-374163-X
ISBN 13: 978-0-13-374163-6

Brief Contents

This page intentionally left blank

Contents

This page intentionally left blank

Preface

Prelude to Programming: Concepts & Design provides a language-independent introduction to programming concepts that helps students learn the following:

- General programming topics, such as data types, control structures, arrays, files, functions, and subprograms
- Structured programming principles, such as modular design, proper program documentation and style, and event-driven and object-oriented program design
- Basic tools and algorithms, such as data validation, defensive programming, sums and averages computation, and searching and sorting algorithms
- Real programming experience through the optional use of RAPTOR, a free flowchart-based programming environment
- Data representation of integer and floating point numbers

No prior computer or programming experience is necessary.

Changes to the Sixth Edition

There are significant and exciting changes in this edition. The text continues to strive to enhance learning programming concepts and to provide students with an enriched experience. Throughout the text, concepts build from clear and simple introductory explanations to complex and challenging Examples and Review Exercises. Major improvements include the following:

- Rather than relegating the material on data representation to Appendices, an entire chapter is devoted to these concepts. This chapter is completely independent of the rest of the content and can be skipped with no loss of continuity. However, instructors who want to include the material now have more examples and end-of-chapter Review Exercises.
- Chapter 0 has been revised with up-to-date content relating to new technologies.
- Chapter 1 has been revised and now includes information on the `Boolean` data type.
- The material on arrays, searching, and sorting has been divided into two chapters. Chapter 7 focuses on creating and using both one- and

two-dimensional arrays. Chapter 8 presents algorithms with extensive examples for searching and sorting.

- The text uses RAPTOR, a free flowcharting software application that allows students to create and run programs without focusing on syntax. Each chapter, from Chapter 3 on, includes an optional section devoted to learning RAPTOR and using RAPTOR to develop interesting, executable programs.
- Throughout the text Examples, Self Checks, and Review Exercises have been redesigned when necessary to ensure that they can be worked with or without RAPTOR.
- The Review Exercises in each chapter contain Multiple Choice, True/False, Short Answer, and a Programming Challenges section. All Challenge problems are suitable for RAPTOR.
- When real code is given throughout the text, JavaScript code has been added.
- More built-in functions and properties are introduced including `Length_Of()`, `To_ASCII()`, `To_Character()`, `Indexing[]`, and more.
- The content in Chapter 11 on object-oriented programming has been thoroughly revised and simplified.
- New material on event-driven programming has been added to Chapter 11.

Organization of the Text

The text is written and organized to allow flexibility in covering topics. Material is presented in such a way that it can be used in any introductory programming course at any level. Each concept is presented in a clear, easily understood manner and the level of difficulty builds slowly. The **What & Why** sidebars give students the opportunity to think above and beyond the material in the Examples and encourage discussion and student interaction. The **Making it Work** sidebars demonstrate how concepts are applied in the real world. **Examples**, **Self Checks**, and **Review Exercises** increase in difficulty from most basic to very challenging. The **Programming Challenges** sections at the end of each chapter give students a chance to create longer, comprehensive programs from scratch and, if RAPTOR is used, they can run the programs and see the results.

The text has been designed so that instructors can use it for students at various levels. The core of the text consists of Chapter 1 and Chapters 3–7. Chapters 0 and 2 are optional; Chapter 2 in particular covers material that is relatively complex and may be skipped without consequence. Chapters 8–11 are independent of one another except that some material in Chapter 9 is required to understand Chapter 11. Thus, the text lends itself to a custom book adoption.

Chapter Summaries

- Chapter 0 provides an overview of general computer concepts.
- Chapter 1 discusses basic problem solving strategy and the essential components of a computer program (input, processing, and output). A section on data types introduces students to numeric, string, and Boolean types.

- Chapter 2 is dedicated to data representation. Students learn to convert decimal numbers to binary and hexadecimal. The various ways to represent integers (unsigned, signed, two's complement) as well as floating point numbers are covered. IEEE standards are used to represent floating point numbers in single- and double-precision. The material in this chapter is completely independent from the rest of the book.
- Chapter 3 introduces the program development process, the principles of modular design, pseudocode, and flowcharts. Documentation, testing, syntax and logic errors, and an overview of the basic control structures are covered.
- Chapter 4 covers decision (selection) structures including single-, dual- and multiple-alternative structures, relational and logical operators, the ASCII coding scheme, defensive programming, and menu-driven programs.
- Chapters 5 and 6 present a complete coverage of repetition structures (loops). Chapter 5 focuses on the basic loop structures: pre- and post-test loops, sentinel-controlled loops, counter-controlled loops, and loops for data input, data validation, and computing sums and averages. Chapter 6 builds on the basics from the previous chapters to create programs that use repetition structures in combination with decision structures, nested loops, and random numbers.
- Chapter 7 covers one-dimensional, two-dimensional, and parallel arrays. Representation of character strings as arrays is also discussed. The material in this chapter has been expanded from the previous edition, including more examples to assist students in understanding this difficult material.
- Chapter 8 covers searching and sorting. Two search techniques (serial and binary searches) and two sort techniques (bubble and selection sorts) are included with expanded coverage.
- Chapter 9 covers functions and modules, including the use of arguments and parameters, value and reference parameters, passing by reference versus passing by value, and the scope of a variable. Built-in and user-defined functions are covered. Recursion—an advanced topic—is discussed in some depth but can be skipped if desired.
- Chapter 10 is about sequential data files. The discussion covers records and fields and how to create, write, and read from sequential files. Topics also include how to delete, modify, and insert records, and how to merge files. Arrays are used in conjunction with data files for file maintenance. The control break processing technique is demonstrated in a longer program.
- Chapter 11 is an introduction to the concepts of object-oriented programming and event-driven programming. The object-oriented material in this chapter has been revised for better understandability. The material on event-driven programming is new to this edition. A short introduction to modeling languages, including UML is given. Object-oriented design topics include classes (parent and child), objects, inheritance, polymorphism, public versus private attributes and methods, and the use of constructors. The material on event-driven programming includes the graphical user interface and window components. Properties and methods for various window controls are also covered.

Many sections throughout the text are devoted to more advanced applications and are optional. In particular, the Focus on Problem Solving sections develop relatively complex program designs, which some instructors may find useful to illustrate the chapter material and others may elect to skip to save time. RAPTOR can be used as a tool to illustrate concepts by creating examples throughout the text in RAPTOR but can also be used to create longer and more challenging, creative programs.

Running With RAPTOR: A Flowcharting Environment

In this edition, each chapter from Chapter 3 onward contains an optional section entitled **Running With RAPTOR**. The section describes how to use RAPTOR for that chapter's material with screenshots and step-by-step instructions. Short examples demonstrate how RAPTOR is used to work with the chapter's content and a longer program is developed. In many chapters the RAPTOR program is an implementation of the long program developed in the Focus on Problem Solving section. The Running With RAPTOR sections can be skipped with no loss of continuity. However, if used, the longer RAPTOR programs give students a real-life experience by creating interesting, running programs including games, encryption, and more.

Features of the Text

In the Everyday World

Beginning with Chapter 1, each chapter starts with a discussion of how the material in that chapter relates to familiar things (for example, "Arrays in the Everyday World") This material provides an introduction to the programming logic used in that chapter through an ordinary and easily understood topic, and establishes a foundation upon which programming concepts are presented.

Making It Work

The **Making It Work** sidebars provide information about how to implement concepts in an actual high-level language, such as C++, Java, JavaScript, or Visual Basic. These boxed sidebars appear throughout the text and are self-contained and optional.

What & Why

Often we conclude an Example with a short discussion about what would happen if the program were run, or what would happen if something in the program were changed. These **What & Why** sidebars help students deepen their understanding of how programs run. They are useful in initiating classroom discussion.

Pointers and Style Pointers

The concepts of programming style and documentation are introduced in Chapter 3 and emphasized throughout. Other **Pointers** appear periodically throughout the text. These short notes provide insight into the subject or specialized knowledge about the topic at hand.

Examples

There are more than 200 numbered worked Examples in the text. The pseudocode in the Examples includes line numbers for easy reference. Detailed line-by-line discussions follow the code with sections entitled **What Happened?**

Focus on Problem Solving

Each chapter from Chapter 4 to the end includes a **Focus on Problem Solving** section which presents a real-life programming problem, analyzes it, designs a program to solve it, discusses appropriate coding considerations, and indicates how the program can be tested. In the process, students not only see a review of the chapter material, but also work through a programming problem of significant difficulty. These sections are particularly useful to prepare students for a language-specific programming course. For selected programs there are real code implementations in C++, Java, Visual Basic, and Python available on the Pearson website which can be used to demonstrate how the concepts learned in the text apply to real-life programs. The program code illustrates the congruence between the pseudocode taught in this book and the code in a specific programming language. Executable files are also included so the actual programs can be run, even if the code is not used pedagogically.

Exercises

Many new exercises have been added to this edition to correspond with new material. Many exercises have been revised to permit them to be implemented with RAPTOR. The text contains the following diverse selection:

- **Self Checks** at the end of each section include items that test students' understanding of the material covered in that section (answers to Self Checks are in Appendix C)
- **Review Questions** at the end of each chapter include questions of various types that provide further review of the chapter material (Answers to the odd-numbered questions are available on the student support website; answers to the even-numbered questions are on the instructor support web site).
- **Programming Challenges** at the end of each chapter require students to design programs using the material learned in that chapter and earlier chapters. All Programming Challenges can be implemented with RAPTOR. Solutions to all Programming Challenges in RAPTOR are available on the instructor support web site.

Supplements

Student Support Web Site

A variety of resources are available with this book. Students may access them at www.pearsonhighered.com/venit-drake.

Instructor's Supplements

Supplemental materials are available to qualified instructors at www.pearsonhighered.com/irc, including the following:

- PowerPoint Presentations for all Chapters
- Solutions to all Self Checks including RAPTOR implementations of select problems
- Solutions to all Review Exercises including corresponding RAPTOR programs
- RAPTOR programs corresponding to all Programming Challenges
- Testbank

For further information about obtaining instructor supplements, contact your campus Pearson Education sales representative.

Acknowledgments

The **In the Everyday World** essays, a unique feature of this book, were envisioned and drafted by Bill Hammerschlag of Brookhaven College for the second edition, and are expanded and revised in this edition.

The implementations of the code in C++, Visual Basic, Java, and Python from the **Focus on Problem Solving** sections were created by Anton Drake from the University of Florida, presently a software developer at OPIE Technologies.

A special thanks to Martin Carlisle who created RAPTOR and remains eager and generous with his support.

We want to extend our thanks to Matt Goldstein, our most supportive and caring Editor; to Marilyn Lloyd, the most patient and understanding Production Manager ever; to Haseen Khan, the Project Manager at Laserwords who works on the other side of the world but feels like my next-door neighbor; and to the entire team at Pearson Education, including Kayla Smith-Tarbox and Yez Alayan. We also want to extend a special thank you to Michael Hirsch who initially brought us together on this project; without Michael, none of this would have been possible.

—Elizabeth Drake
and Stewart Venit

I want to thank my coauthor, Stewart Venit. It's a pleasure to work with him. Marilyn Lloyd and Haseen Khan are very special people; they answer my questions with unfailing patience. I also want to thank my children, Anton and Severia, who

have always encouraged my desire—my need—to write. My grandsons, Justy and Jacob, make me smile by being impressed by my work.

—Elizabeth Drake

I would like to thank my coauthor, Elizabeth Drake, for greatly enhancing and improving this book in each of the last four editions. I am grateful to my wife Corinne, who, over the course of my 35 year writing career, never complained about the countless hours I spent camped in front of a computer screen. I also want to thank the rest of my family for being my family: daughter Tamara, son-in-law Cameron, and grandchildren Evelyn and Damian.

—Stewart Venit

This page intentionally left blank

Introduction

0

In this introduction, we will discuss the history of computers and computer **hardware** and **software**—the devices and programs that make a computer work.

After reading this introduction, you will be able to do the following:

- Understand the evolution of computing devices from ancient Babylonia to the twenty-first century
- Understand the components that make up a typical computer system: the central processing unit, internal memory, mass storage, and input and output devices
- Know the types of internal memory—RAM and ROM—and understand their functions
- Know the types of mass storage: magnetic, optical, solid state, and online storage
- Know the types of software used by a modern computer: application software and system software
- Know the levels of programming languages: machine language, assembly language, and high-level language
- Know the types of programming and scripting languages used to create software
- Understand the distinction between programming and scripting languages

In the Everyday World

Computers Everywhere

A century ago, a child would listen in wonder as his parents described what life was like before cars, electricity, and telephones. Today, a child listens in wonder as his parents describe what life was like without video games, smart phones, GPS systems, and computers. Seventy years ago, electronic computers didn't exist. Now, we use computers daily. Computers are in homes, schools, and offices; in supermarkets and fast food restaurants; on airplanes and submarines. Computers are in our phones, kitchen appliances, and cars. We carry them in our backpacks, pockets, and purses. They are used by the young and old, filmmakers and farmers, bankers and baseball managers. By taking advantage of a wealth of diverse and sophisticated software (programs and apps), we are able to use computers almost limitlessly for education, communication, entertainment, money management, product design and manufacture, and business and institutional processes.

0.1 A Brief History of Computers

Calculators, devices used to increase the speed and accuracy of numerical computations, have been around for a long time. For example, the abacus, which uses rows of sliding beads to perform arithmetic operations, has roots that date back more than 5,000 years to ancient Babylonia. More modern mechanical calculators, using gears and rods, have been in use for almost 400 years. In fact, by the late nineteenth century, calculators of one sort or another were relatively commonplace. However, these machines were by no means *computers* as we use the word today.

What Is a Computer?

A **computer** is a mechanical or an electronic device that can efficiently store, retrieve, and manipulate large amounts of information at high speed and with great accuracy. Moreover, it can execute tasks and act upon intermediate results without human intervention by carrying out a list of instructions called a **program**.

Although we tend to think of the computer as a recent development, Charles Babbage, an Englishman, designed and partially built a true computer in the mid-1800s. Babbage's machine, which he called an *Analytical Engine*, contained hundreds of axles and gears and could store and process 40-digit numbers. Babbage was assisted in his work by Ada Augusta Byron, the daughter of the poet Lord Byron. Ada Byron grasped the importance of the invention and helped to publicize the project. A major programming language (Ada) was named after her. Unfortunately, Babbage never finished his Analytical Engine. His ideas were too advanced for the existing technology, and he could not obtain enough financial backing to complete the project.

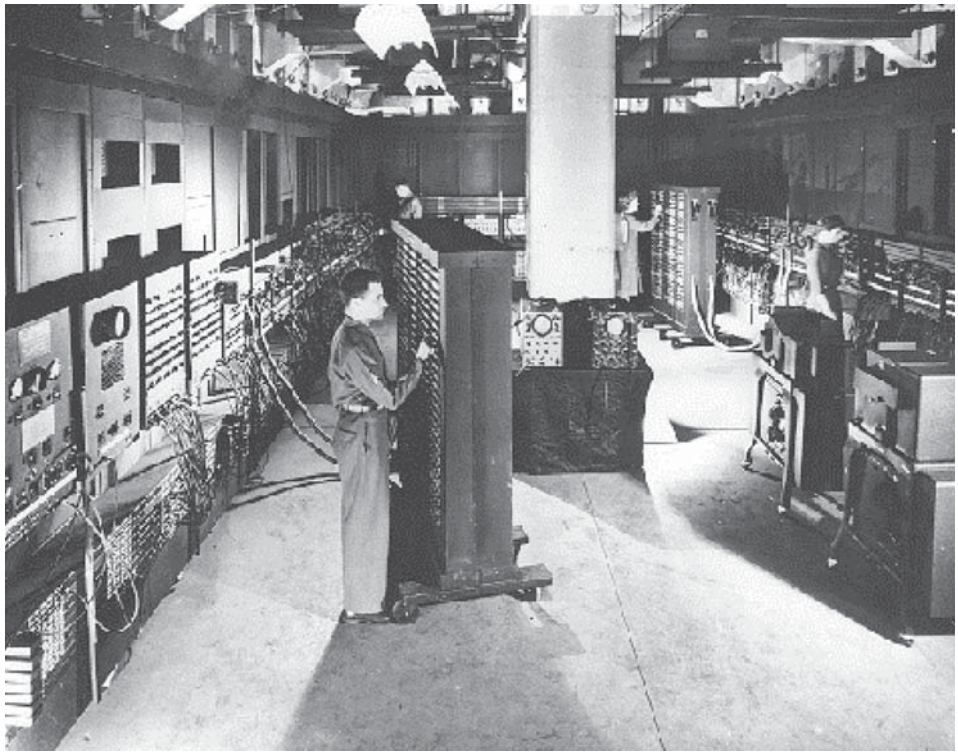
Serious attempts to build a computer were not renewed until nearly 70 years after Babbage's death. Around 1940, Howard Aiken at Harvard University, John Atanasoff, and Clifford Berry at Iowa State University built machines that came close to being true computers. However, Aiken's Mark I could not act independently on

its intermediate results, and the Atanasoff-Berry computer required the frequent intervention of an operator during its computations.

Just a few years later in 1945, a team at the University of Pennsylvania, led by John Mauchly and J. Presper Eckert, completed work on the world's first fully operable electronic computer. Mauchly and Eckert named it ENIAC, an acronym for Electronic Numerical Integrator and Computer. ENIAC (see Figure 0.1) was a huge machine. It was 80 feet long, 8 feet high, weighed 33 tons, contained over 17,000 vacuum tubes in its electronic circuits, and consumed 175,000 watts of electricity. For its time, ENIAC was a truly amazing machine because it could accurately perform up to 5,000 additions per second. However, by current standards, it was exceedingly slow. A modern run-of-the-mill personal computer can exceed 100 million operations per second!

For the next decade or so, all electronic computers used **vacuum tubes** (see Figure 0.2) to do the internal switching necessary to perform computations. These machines, which we now refer to as first-generation computers, were large by modern standards, although not as large as ENIAC. They required a climate-controlled environment and a lot of tender love and care to keep them operating. By 1955, about 300 computers—built mostly by IBM and Remington Rand—were being used, primarily by large businesses, universities, and government agencies.

Figure 0.1 The ENIAC computer



Source: U.S. Army

Figure 0.2 A vacuum tube

By the late 1950s, computers had become much faster and more reliable. The most significant change at this time was that the large, heat-producing vacuum tubes were replaced by relatively small transistors. The **transistor** (see Figure 0.3) is one of the most important inventions of the twentieth century. It was developed at Bell Labs in the late 1940s by William Shockley, John Bardeen, and Walter Brattain, who later shared a Nobel Prize for their achievement. Transistors are small and require very little energy, especially compared to vacuum tubes. Therefore, many transistors can be packed close together in a compact enclosure.

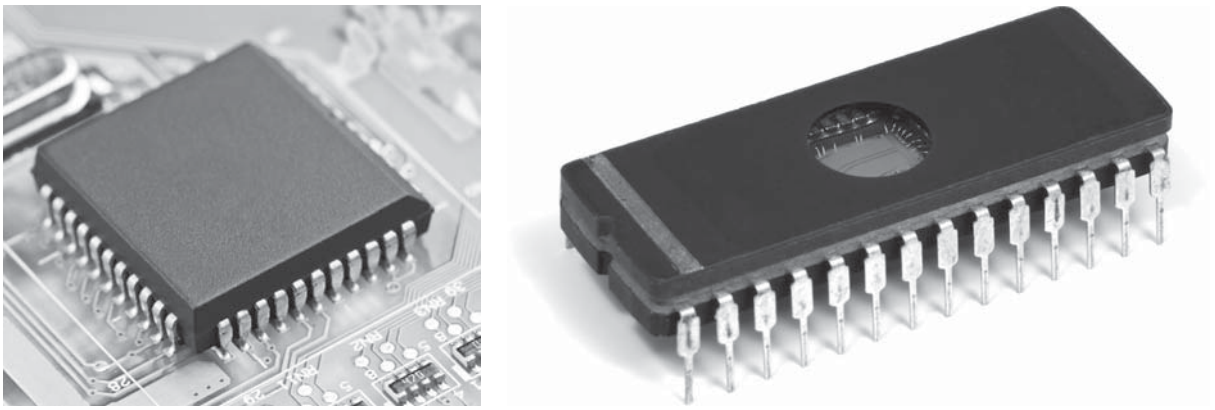
In the early 1960s, Digital Equipment Corporation (DEC) took advantage of small, efficient packages of transistors called **integrated circuits** to create the **minicomputer**, a machine roughly the size of a four-drawer filing cabinet. Because these computers not only were smaller but also less expensive than their predecessors, they were an immediate success. Nevertheless, sales of larger computers, now called **mainframes**, also rapidly increased. The computer age had clearly arrived and the industry leader was the IBM innovative System 360.

Personal Computers

Despite the increasing popularity of computers, it was not until the late 1970s that the computer became a household appliance. This development was made possible by the invention of the **microchip** (see Figure 0.4) in the 1960s. A microchip is a piece of silicon about the size of a postage stamp, packed with thousands of electronic components. The microchip and its more advanced cousin, the **microprocessor**, led to the creation of the world's first **personal computer (PC)** in 1974. The PC

Figure 0.3 An early transistor

Figure 0.4 The microchip



was relatively inexpensive compared to its predecessors and was small enough to fit on a desktop. This landmark computer, the Altair 8800 microcomputer, was unveiled in 1975. Although it was a primitive and not a very useful machine, the Altair inspired thousands of people, both hobbyists and professionals to become interested in PCs. Among these pioneers were Bill Gates and Paul Allen, who later founded Microsoft Corporation, now one of the world's largest companies.

Apple Computers and the IBM PC

The Altair also captured the imagination of two young Californians, Stephen Wozniak and Steven Jobs. They were determined to build a better, more useful computer. They founded Apple Computer, Inc., and in 1977 they introduced the Apple II, which was an immediate hit. With the overwhelming success of this machine and Tandy Corporation's TRS-80, companies that were manufacturing larger minicomputers and mainframes began to notice. In 1981, IBM introduced the popular IBM PC (see Figure 0.5), and the future of the PC was assured.

Figure 0.5 The IBM PC, introduced in 1981, is an antique now!

Many companies hoping to benefit from the success of the IBM PC, introduced computers that could run the same programs as the IBM, and these "IBM compatibles" soon dominated the market. Even the introduction of Apple's innovative and easy-to-use Macintosh in 1984 could not stem the tide of the IBM compatibles. These computers, virtually all of which make use of Microsoft's Windows operating system, have also spawned a huge array of software (computer programs) never dreamed of by the manufacturers of the original mainframes. This software includes word processors, photo editing programs, Web browsers, spreadsheet programs, database systems, presentation graphics programs, and a seemingly infinite variety of computer games. However, while in 2000 the Windows operating system commanded more than 95% of the market share, today's mobile devices, such as smart phones and tablets, have reduced Microsoft's domination drastically with Google's Android operating system and the Apple operating system providing strong competition.

Today's Computers

Today the computer market comprises a vast array of machines. Personal computers are everywhere and range in price from a few hundred to a few thousand dollars. For the most part, their manufacturers are billion dollar companies like IBM, Dell, Hewlett-Packard, and Apple. Although PCs are small and inexpensive, they produce a remarkable amount of computing power. Today's tablets, which can weigh less than a pound and fit into a handbag, are far more powerful than the most advanced mainframes of the mid-1970s (see Figure 0.6).

Minicomputers have also found their niche. Unlike PCs, these machines can be used by a number of people (typically 16 or more) working simultaneously at separate and remote **terminals**. Each terminal consists of a keyboard and a display screen. Minicomputers have become the mainstay of many small businesses and universities, but mainframe computers are by no means dinosaurs. These relatively large and costly machines supply users with tremendous power to manipulate information. **Supercomputers** (see Figure 0.7) are even more powerful than mainframes and can process well over 1 billion instructions per second. For a special effects company like Industrial Light and Magic or a government agency like the Internal Revenue Service, there is no substitute for a large mainframe or supercomputer.

Figure 0.6 Today's laptop and tablet computers

Figure 0.7 The Jaguar/Cray XT5—a modern supercomputer

The Internet

Despite the recent advances in computer technology, arguably the most significant development in the last 15 years has been the phenomenal rise in popularity of the **Internet**—a worldwide collection of *networks*. A **network** consists of two or more linked computers that are able to share resources and data wirelessly or via cable or phone lines. The Internet has roots that date back to a relatively small U.S. Defense Department project in the late 1960s. Since then, the Internet has grown from a small collection of mainframe computers used by universities and the military to more than 2.5 billion users worldwide who range in age from preschoolers to centenarians. Before the advent of smart phones, the two main attractions of the Internet were email and the World Wide Web. **Email**, which is short for *electronic mail*, allows anyone with access to the Internet to use his or her computer to exchange messages with another Internet user anywhere in the world almost instantaneously and at little or no cost. The **World Wide Web** (more simply called the **Web**) originated in 1989. It is a vast collection of linked documents (web pages) created by Internet users and stored on thousands of Internet-connected computers.

Today social networking sites have rivaled the popularity of email. These sites are part of Web2.0, the next generation of the World Wide Web. **Web2.0** consists of web applications that facilitate information sharing, user-centered design, and collaboration. While the term suggests a new version of the Web, it is not an update or change in technical specifications, but rather a change in the way people use the Web. Web2.0 generally refers to Web-based communities (such as Second Life), wikis (such as the online encyclopedia, Wikipedia), social-networking sites (such as Facebook), video-sharing sites (like YouTube), and more.

Self Check for Section 0.1

0.1 What characteristics of a computer distinguish it from the following?
   a. A simple (non-programmable) calculator
   b. A programmable calculator

0.2 Complete each of the following statements:
   a. A _____ is a list of instructions to be carried out by the computer.
   b. The first fully operative electronic computer was called _____.
   c. The fastest computers in use today are called _____.
   d. The _____ is a worldwide collection of interlinked networks.

0.3 True or false? The first personal computers were produced in the 1970s.

0.4 True or false? Transistors, which eventually replaced the vacuum tubes of early computers, were invented in the 1940s.

0.5 True or false? Minicomputers and mainframe computers have become obsolete; they are no longer in use.

0.6 True or false? Web2.0 is a new version of the World Wide Web, including updated technical specifications.

0.7 Match the people listed in the first column with the corresponding computer from the second column.

1. Charles Babbage and Ada Byron	___	a. ENIAC
2. J. Presper Eckert and John Mauchly	___	b. Apple II
3. Steven Jobs and Stephen Wozniak	___	c. Analytical Engine

0.2 Computer Basics

In Section 0.1 we defined a computer as a mechanical or an electronic device that can efficiently store, retrieve, and manipulate large amounts of information at high speed and with great accuracy. Regardless of the type of computer—from the original desktop machine with a separate tower, monitor, keyboard, and mouse to the smallest, thinnest smart phone to the elaborate digital display on a car's dashboard—a computer consists of the same basic components. As the definition implies, a computer must have the ability to input, store, manipulate, and output data. These functions are carried out by the following five main components of a computer system:

1. The central processing unit (CPU)
2. Internal memory (consisting of RAM and ROM)
3. Mass storage devices (magnetic, optical, and solid state) and the Cloud
4. Input devices (primarily the keyboard and mouse)
5. Output devices (primarily the monitor and printer)

In this section, we will describe these components as they are implemented on a modern personal computer.

In a desktop PC, the CPU, internal memory, and most mass storage devices are located in the **system unit**. The input and output devices are housed in their own enclosures and are connected to the system unit by cables, or more recently, by wireless transmitters. Components like these, which are used by a computer but located outside the system unit, are sometimes referred to as **peripherals**. Laptop and tablet computers house the CPU, internal memory, mass storage devices, a monitor, and a keyboard, all in one relatively small package. The physical equipment that makes up the computer system is known as hardware.

The Central Processing Unit

The **central processing unit** (also called the **processor** or **CPU**) is the brain of the computer. It receives the program instructions, performs the arithmetic and logical operations necessary to execute them, and controls the other computer components. In a PC, the processor consists of millions of transistors that reside on a single microchip about the size of a postage stamp and plug into the computer's main circuit board, the **motherboard**.

More than any other component, the CPU distinguishes one computer from another. A primary factor in determining the power of a processor is its speed, measured in *gigahertz* (GHz). For example, the Pentium IV microprocessor, produced by Intel Corporation for use in PCs, is a chip that is produced in several variations that run at speeds up to 4 GHz. However, due to various factors, some processors are more powerful than others running at the same speed.

Internal Memory

A computer uses its **internal memory** to store instructions and data to be processed by the CPU. In a PC, memory resides on a series of chips either plugged directly into the motherboard or into one or more smaller circuit boards connected to the motherboard. There are two types of internal memory: **read-only memory (ROM)** and **random-access memory (RAM)**.

ROM contains an unalterable set of instructions that the computer uses during its start-up process and for certain other basic operations. RAM on the other hand, can be read from and written to. It's used by the computer to hold program instructions and data. You can think of ROM as a reference sheet, and RAM as a scratchpad, albeit a very large scratchpad. ROM is an integrated circuit programmed with specific data when it is manufactured. This information cannot be altered by the user; therefore, ROM is a permanent form of memory storage while RAM is the memory used by the computer to hold the data you are working on at any given time. For example, if you are writing an English essay with a word processor, as you write the essay you see it on your monitor, but it is also being held in RAM. When you close the word processing program or turn off the computer, all the information stored in RAM is lost. That's why it's important to save your work to some permanent storage medium—as most of us have learned at one time or another, to our dismay!

The smallest unit of memory is the **bit**. A bit can store only two different values—a zero or a one. It takes eight bits—one byte—to store a single character in memory. (Loosely speaking, a *character* is any symbol you can type, such as a letter, digit, or a punctuation mark.) Storing an instruction usually takes sixteen or more bits.

It would be impractical to talk about the size of a file in bits when just a single line of text might use hundreds of bits. Instead, the basic unit of memory is a **byte**. Memory is measured in **kilobytes (KB)**, **megabytes (MB)**, or **gigabytes (GB)**. One kilobyte is 1,024 ($1{,}024 = 2^{10}$) bytes and one megabyte is 1,024 kilobytes. A gigabyte is 1,073,741,824 ($1{,}024^9$ or 2^{30}) bytes. For example, 128 MB of RAM can store 134,217,728 characters of information at one time because, mathematically, the number of characters equals the number of bytes as follows:

$$128 \text{ MB} \times \frac{1{,}024 \text{ KB}}{\text{MB}} \times \frac{1{,}024 \text{ bytes}}{\text{KB}} = 134{,}217{,}728 \text{ bytes}$$