Preface to Third Edition

This text has two primary objectives: to teach the basic principles of programming and to teach the basic constructs of the C Language. Our text puts these objectives in the context of good software engineering concepts that we have developed through more than 30 years of experience in industry and academia.

Major Changes in Third Edition

This is a major change from the second edition. In addition to significant changes in the presentation and material covered, it also updates the language to the ISO/IEC 9899 (1999), or as it is more commonly known, C99.

While we have made major changes, we have held true to our primary objectives as outlined above. The text remains a comprehensive introduction to computer programming in a software engineering context.

The major changes are outlined in the following sections.

Standard C (1999)

The following topics, in alphabetical order, are explicitly discussed in the text.

- the Boolean type
- complex arithmetic
- *for* statement declarations
- line comments
- character set extensions (including Unicode and wide-characters)
- extended math library
- integer type extensions

A C Language Perspective

While C is a complex and professional language, our classroom experience using the previous editions of this book has shown that beginning students can easily understand it. We believe that if the language is put into a perspective that allows the student to understand its design, C is not a difficult language.

There are two aspects of C that separate it from most other languages: expressions and pointers. The concept of expressions, as used in C, is unique. The first half of this text builds a firm understanding of expressions. We introduce pointers only to the extent necessary to cover passing them to functions in Chapter 7 and arrays in Chapter 8. Our experiences have shown that with a firm grasp of the expression concept, much of the mystery of C disappears.

In Chapters 9, we begin to develop the concept of pointers, which are further developed in Chapter 10. Chapter 11 provides a discussion of C's approach to strings.

The last chapter is a simple introduction to data structures. While not all courses will have time to cover this chapter, those that do will give the students a head start into data structures. Motivated students will find that they can "get a leg up" on data structures over the term break.

The appendices at the end of the text comprise a rich set of references to subjects required in the complete C language. These will be of use to students who go on to take other courses in the C language.

Features of the Book

Several features of this book make it unique and easy for beginning students to understand.

Structure and Style

One of our basic tenets is that good habits are formed early. The corollary is that bad habits are hard to break. Therefore, we consistently emphasize the principles of structured programming and software engineering. Throughout each chapter, short code snippets and programs are used to demonstrate techniques.

Complete programs, generally found in the last section of the chapter, are written in a well-document consistent style. As programs are analyzed, style and standards are further explained. While we acknowledge that there are many good styles, our experience has shown that if students are exposed to one good style and adopt it, they will be better able to adapt to other good styles.

Principle Before Practice

Whenever possible, we develop the principle of a subject before we introduce the language implementation. For example, in Chapter 6 we first introduce the concept of logical data and selection and then we introduce the *if . . . else* and *switch* statements. This approach gives the student an understanding of selection before introducing the nuances of the language.

Visual Approach

A brief scan of the book will demonstrate that our approach is visual. There are more than 400 figures, plus more than 70 tables and 180 program examples. While this amount of material tends to create a large book, the visual approach makes it easy for students to follow the material.

Examples

While the programming examples vary in complexity, each uses a consistent style. Our experience working with productional programs that live for 10 to 20 years convinced us that readable and understandable programs are easier to work with than programs written in a terse, cryptic manner. For that reason, and to emphasize the structure of the language, we label the sections in a function with comments. We consistently follow a style that places only one declaration, definition, or statement on a line.

Coding Techniques

Throughout the text we include coding techniques that make programs more readable and often more efficient. For example, in the analylsis of Program 6-4 you will find the following discussion:

Where do we check for *greater than*? The answer is that we default the *greater than* condition to the outer *else*. . . . When coding a two-way selection statement, try to code the most probable condition first; with nested selection statements, code the most probable first and the least probable last.

These techniques are drawn from our extensive industry and classroom experience.

Software Engineering

A discussion of software engineering principles concludes each chapter. Our intent is not to replace a separate course in software engineering. Rather, we firmly believe that by incorporating basic software engineering principles early in their studies, students will be better prepared for a formal treatment of the subject. Even more important, by writing well-engineered programs from the beginning, students will not be forced to unlearn and relearn. They will better understand software discussions in their subsequent classes.

While the software engineering sections are found at the end of each chapter, they are most successfully taught by introducing them as the chapter unfolds. Then, a short review at the end of the chapter summarizes the principles that have been demonstrated during the lectures.

These sections are visually distinguishable from the rest of the chapter. They have been set apart for several reasons. First, they are reality a small book within a book. While these sections contain important material, the book stands on its own without them. You ma therefore, decide to cover the software engineering sections as formal lecture topics or informally while the chapter material is being co ered. You may decide to assign them to the student as additional reading, or, you may decide to exclude them entirely from the course.

In general, software engineering sections directly or indirectly pertain to the chapter material. Where they don't, they discuss gene software engineering subjects, such as cohesion, coupling, and quality.

Pedagogical Material

End chapter material meets two pedagogical objectives: first, it helps the students to review or summarize what they have learned, a second, it tests the students' mastery of the chapter material.

Review Material

- **Tips and Common Programming Errors** points out helpful hints and possible problem areas.
- **Key Terms** provides a list of the important terms introduced in the chapter.
- **Summary** contains a concise overview of the key points for students to understand in the chapter.

Practice Sets

- **Review Questions** contain several multiple choice questions that are similar to the questions found in the examination databa The answers to the odd-numbered questions are included in the solutions on the Course Technology web site.
- **Exercises** are short questions covering the material in the chapter. The answers to the odd-numbered exercises are also included the solutions on the Course Technology web site.
- **Problems** are short coding problems, generally intended to be run on a computer. They can usually be developed in two to thr hours. Once again, odd-numbered solutions are found on the web site.
- **Projects** are longer, major assignments that may take the average student six to nine hours to develop.

Appendices and Cover Material

The appendices are intended to provide quick reference material, such as the Unicode Character Set, or to provide a review of materi such as numbering systems, usually covered in a general computer class.

The inside covers contain two important references that are used continually throughout the course—the Precedence Table and th Formatted I/O Codes.

Acknowledgments

No book of this scope can be developed without the support of many people.

Reviewers

To anyone who has not been through the process, the value of peer reviews cannot be appreciated enough. Writing a text rapidly becom a myopic process. The important guidance of reviewers who can stand back and review the text as a whole cannot be measured. W would especially like to acknowledge the contributions of the reviewers of all three editions.

Stephen Allen, *Utah State University*
Ali Behforooz, *Towson State University*
Ernest Carey, *Utah Valley State College*
Constance Conner, *City College of San Francisco*
Maurice L. Eggen, *Trinity University*
Robert Gann, *Hartwick College*
Rick Graziani, *Cabrillo College*
Jerzy Jaromczyk, *University of Kentucky*
Roberta Klibaner, *College of Staten Island*
Krishna Kulkarni, *Rust College*
Mike Michaelson, *Palomar College*
Ali Nikzad, *Huston-Tillotson College*
Mark Parker, *Shoreline Community College*
Oskar Reiksts, *Kutztown University*
Ali Salenia, *South Dakota State University*
Shashi Shekhar, *University of Minnesota*
Brenda Sonderegger, *Montana State University*
Venkat Subramanian, *University of Houston*
Marc Thomasm, *California State University, Bakersfield*
K.C. Wong, *Fayetteville State University*
Mary Astone, *Troy State University*
George Berry, *Wentworth Institute of Technology*
Ping Chu Chu, *Fayetteville State University*
John S. DaPonte, *Southern Connecticut State University*
Peter Gabrovsky, *CSU Northridge*
Henry Gordon, *Kutztown University*.
Barbara Guillott, *Louisiana State University*
John Kinio, *Humber College*
Joseph A. Konstan, *University of Minnesota*
John Lowther, *Michigan Technological University*
Kara Nance, *University of Alaska—Fairbanks*
Jo Ann Parikh, *Southern Connecticut State University*
Savitha Pinnepalli, *Louisiana State University*
Jim Roberts, *Carnegie Mellon University*
Larry Sells, *Oklahoma City University*
Robert Signorile, *Boston College*
Deborah Sturm, *College of Staten Island*
John B. Tappen, *University of Southern Colorado*
John Trono, *St. Michael's College*

Course Technology Staff

Our thanks to our editors, Alyssa Pratt, Senior Product Manager, and Mary Franz, Senior Acquisitions Editor, for helping us produce th book. We are also indebted to the Quality Assurance staff who diligently double-checked each chapter and program. Our thanks to Bu LaFountain, Serge Palladino, and Chris Scriver.

We would also like to thank the staff at GEX Publishing Services, especially Sandra Mitchell, who ably guided the book through pr duction.

Behrouz A. Forou
Richard F. Gilb

Scanned by CamScanner

Chapter 1

Introduction to Computers

Welcome to computer science! You are about to explore a wonderful and exciting world—a world that offers many challenging and exciting careers.

In this chapter, we introduce you to the concepts of computer science, especially as they pertain to computer programming. You will study the concept of a computer system and how it relates to computer hardware and software. We will also present a short history of computer programming languages so that you understand how they have evolved and how the C language fits into the picture.

We will then describe how to write a program, first with a review of the tools and steps involved, and then with a review of a system development methodology.

Objectives

- ❑ To review basic computer systems concepts
- ❑ To be able to understand the different computing environments and their components
- ❑ To review the history of computer languages
- ❑ To be able to list and describe the classifications of computer languages
- ❑ To understand the steps in the development of a computer program
- ❑ To review the system development life cycle

1.1 COMPUTER SYSTEMS

Today computer systems are found everywhere. Computers have become almost as common as televisions. But what is a computer? A computer is a system made of two major components: hardware and software. The computer hardware is the physical equipment. The software is the collection of programs (instructions) that allow the hardware to do its job. Figure 1-1 represents a **computer system**.

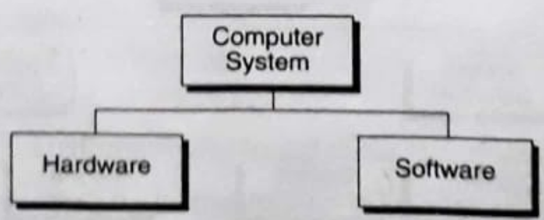

Figure 1-1 A Computer System

Computer Hardware

The **hardware** component of the computer system consists of five parts: input devices, central proce ing unit (CPU), primary storage, output devices, and auxiliary storage devices (Figure 1-2).

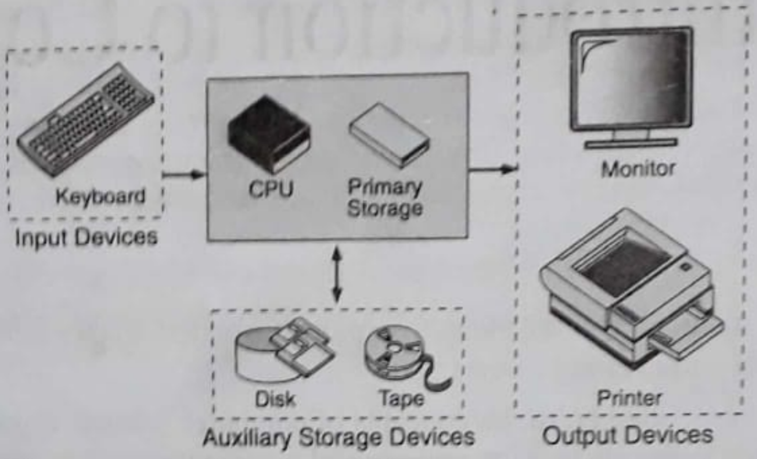

Figure 1-2 Basic Hardware Components

The **input device** is usually a keyboard where programs and data are entered into the comput Examples of other input devices include a mouse, a pen or stylus, a touch screen, or an audio input un

The **central processing unit (CPU)** is responsible for executing instructions such as arithmetic c culations, comparisons among data, and movement of data inside the system. Today's computers m have one, two, or more CPUs. **Primary storage**, also known as **main memory**, is a place where the p grams and data are stored temporarily during processing. The data in primary storage are erased wh we turn off a personal computer or when we log off from a time-sharing computer.

The **output device** is usually a monitor or a printer to show output. If the output is shown on t monitor, we say we have a **soft copy**. If it is printed on the printer, we say we have a **hard copy**.

Auxiliary storage, also known as **secondary storage**, is used for both input and output. It is the pla where the programs and data are stored permanently. When we turn off the computer, our progra and data remain in the secondary storage, ready for the next time we need them.

Computer Software

Computer **software** is divided into two broad categories: system software and application softwa This is true regardless of the hardware system architecture. System software manages the compu resources. It provides the interface between the hardware and the users but does nothing to direc serve the users' needs. Application software, on the other hand, is directly responsible for helping use solve their problems. Figure 1-3 shows this breakdown of computer software.

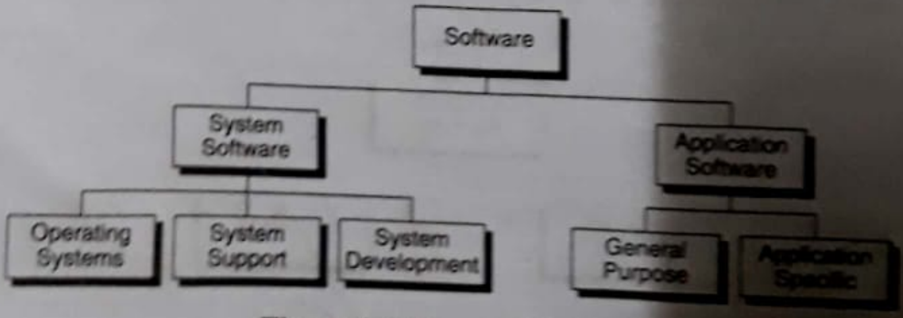

Figure 1-3 Types of Software

System Software

System software consists of programs that manage the hardware resources of a computer and perform required information processing tasks. These programs are divided into three classes: the operating system, system support, and system development.

The **operating system** provides services such as a user interface, file and database access, and interfaces to communication systems such as Internet protocols. The primary purpose of this software is to keep the system operating in an efficient manner while allowing the users access to the system.

System support software provides system utilities and other operating services. Examples of system utilities are sort programs and disk format programs. Operating services consist of programs that provide performance statistics for the operational staff and security monitors to protect the system and data.

The last system software category, **system development software**, includes the language translators that convert programs into machine language for execution, debugging tools to ensure that the programs are error-free, and computer-assisted software engineering (CASE) systems that are beyond the scope of this book.

Application Software

Application software is broken into two classes: general-purpose software and application-specific software. **General-purpose software** is purchased from a software developer and can be used for more than one application. Examples of general-purpose software include word processors, database management systems, and computer-aided design systems. They are labeled general purpose because they can solve a variety of user computing problems.

Application-specific software can be used only for its intended purpose. A general ledger system used by accountants and a material requirements planning system used by a manufacturing organization are examples of application-specific software. They can be used only for the task for which they were designed; they cannot be used for other generalized tasks.

The relationship between system and application software is seen in Figure 1-4. In this figure, each circle represents an interface point. The inner core is the hardware. The user is represented by the outer layer. To work with the system, the typical user uses some form of application software. The application software in turn interacts with the operating system, which is a part of the system software layer. The system software provides the direct interaction with the hardware. Note the opening at the bottom of the figure. This is the path followed by the user who interacts directly with the operating system when necessary.

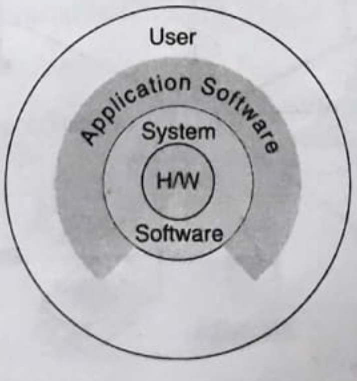

Figure 1-4 Relationship Between System and Application Software

If users cannot buy software that supports their needs, then a custom-developed application must be built. In today's computing environment, one of the tools used to develop software is the C language that we will be studying in this text.

1.2 COMPUTING ENVIRONMENTS

In the early days of computers, there was only one environment: the mainframe computer hidden i central computing department. With the advent of minicomputers and personal computers, the envir ment changed, resulting in computers on virtually every desktop. In this section we describe seve different environments.

Personal Computing Environment

In 1971, Marcian E. Hoff, working for Intel, combined the basic elements of the central processing into the microprocessor. This first computer on a chip was the Intel 4004 and was the grandpar many times removed of Intel's current system.

If we are using a personal computer, all of the computer hardware components are tied together our **personal computer** (or **PC**[1] for short). In this situation, we have the whole computer for ourself; can do whatever we want. A typical personal computer is shown in Figure 1-5.

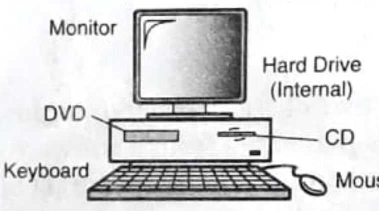

Figure 1-5 Personal Computing Environment

12

Time-Sharing Environment

Employees in large companies often work in what is known as a **time-sharing environment**. In th time-sharing environment, many users are connected to one or more computers. These computers m be minicomputers or central mainframes. The terminals they use are often nonprogrammable, althou today we see more and more microcomputers being used to simulate terminals. Also, in the time-sha ing environment, the output devices (such as printers) and auxiliary storage devices (such as disks) a shared by all of the users. A typical college lab in which a minicomputer is shared by many students shown in Figure 1-6.

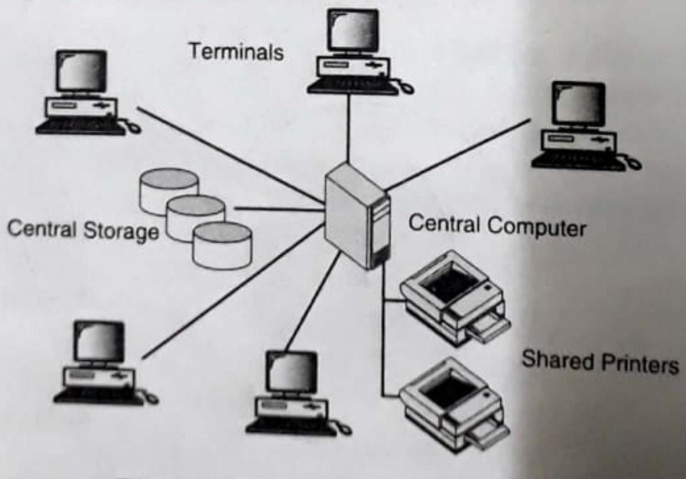

Figure 1-6 Time-sharing Environment

1. PC is now generally accepted to mean any hardware using one of Microsoft's Windows operating systems, as opposed to Apple's Macintosh. We use it in the original, generic sense meaning any personal computer.