ANSI C

Structured Programming • Computer Concepts • Programming Languages • History of C • C Programming Environment • Memory Concepts • Decision Making • Algorithms • Pseudocode • Software Engineering • Control Structures • Top-down, Stepwise Refinement • Program Control • Sequence • Selection • Repetition • Functions • Standard Library Functions • Function Prototypes • Call by Value • Call by Reference • Random Number Generation • Iteration • Recursion • Arrays • Sorting • Searching • Pointers • Pointer Arithmetic • Function Pointers • String Processing • Character Processing • Standard Input/Output • Structures • Unions • Bit Manipulation • Enumerations • Sequential Files • Random Access Files • Data Structures • Dynamic Memory Allocation • Lists • Queues • Stacks • Trees • Preprocessor • Conditional Compilation • C++ • Concurrent C • Object-Oriented Programming • Data Abstraction • Software Reusability • Stream Input/Output • Inline Functions • Reference Parameters • Classes • Objects • Member Functions • Base Classes • Inheritance • Derived Classes • Overloading • Polymorphism • Virtual Functions • Number Systems • Objectives • Quotes • Outlines • Summaries • Terminology • Common Programming Errors • Good Programming Practices • Performance Tips • Portability Tips • Self-Review Exercises and Answers • Exercises

# HOW TO PROGRAM

H. M. Deitel / P. J. Deitel

# C How to Program

H. M. Deitel
Nova University
Deitel and Associates

P. J. Deitel
Deitel and Associates



PRENTICE HALL, Englewood Cliffs, New Jersey 07632

*Library of Congress Cataloging in Publication Data*

Deitel, Harvey M.
C how to program / H.M. Deitel, P.J. Deitel.
p. cm.
Includes bibliographical references and index.
ISBN 0-13-118043-6
1. C (Computer program language) I. Deitel, Paul J. II. Title.
QA76.73.C15D44 1992
005.13'3--dc20 91-46289
CIP

Acquisitions editor: ***Marcia Horton***
Editorial production/supervisor: ***Kathleen Schiaparelli***
Cover design: ***Wanda Lubelska***
Prepress buyer: ***Linda Behrens***
Manufacturing buyer: ***Dave Dickey***
Supplements editor: ***Alice Dworkin***
Editorial assistant: ***Diana Penha***





Printed in the United States of America

10 9 8 7 6 5 4 3 2

ISBN 0-13-118043-6

Prentice-Hall International (UK) Limited, *London*
Prentice-Hall of Australia Pty. Limited, *Sydney*
Prentice-Hall Canada Inc., *Toronto*
Prentice-Hall Hispanoamericana, S.A., *Mexico*
Prentice-Hall of India Private Limited, *New Delhi*
Prentice-Hall of Japan, Inc., *Tokyo*
Simon & Schuster of Asia Pte. Ltd., *Singapore*
Editora Prentice-Hall do Brasil, Ltda., *Rio de Janerio*

*To*

*Dr. Abraham Fischler, President of Nova University and to*
*Dr. Edward Simco, Dean of the Nova University Center for Computer and Information Sciences*

*For their vision of an institution for advanced scientific and computer science education and research in South Florida, and for their indefatigable efforts in realizing that vision in this young university.*

*H. M. D.*

*To*

*My teachers at Lawrenceville and M.I.T.*

*For instilling in me a love of learning and writing.*

*P. J. D.*

# Preface

Welcome to C! This book is by an old guy and a young guy. The old guy (HMD; Massachusetts Institute of Technology 1967) has been programming and/or teaching programming for more than 30 years. The young guy (PJD; MIT 1991) has been programming for a dozen years and has caught the teaching and writing "bug." The old guy programs and teaches from experience. The young guy programs from an inexhaustible reserve of energy. The old guy wants clarity. The young guy wants performance. The old guy appreciates elegance and beauty. The young guy wants results. We got together to produce a book we hope you will find informative, interesting, and entertaining.

In most educational environments, C is taught to people who already know how to program. Many educators believe that the complexity of C, and a number of other difficulties, make C unworthy for a first programming course—precisely the target course for this book. So why did we write this text?

C has in fact become the systems implementation language of choice in industry, and there is good reason to believe that its object-oriented variant, C++, will emerge as the dominant language of the mid-to-late 1990s. Harvey Deitel has been teaching Pascal in university environments for 13 years with an emphasis on developing clearly written, well-structured programs. Much of what is taught in an introductory Pascal course sequence is the basic principles of structured programming. We have presented this material exactly the way HMD has done in his university courses. There are some pitfalls, but where these occur, we point them out and explain procedures for dealing with them effectively. Our experience has been that students handle the course in about the same manner as they handle Pascal. There is one noticeable difference though: The students are highly motivated by the fact that they are learning a language that will be immediately useful to them as they leave the university environment. This increases their enthusiasm for the material—a big help when you consider that C is considerably more difficult to learn.

Our goal was clear: Produce a programming textbook using C for introductory university-level courses in computer programming for students with little or no programming experience. But produce a book that also offers the rigorous treatment of theory and practice demanded by traditional C programming courses. To meet these goals, we produced a book larger than other C texts—this because our text also patiently teaches structured programming principles. Approximately 1000 students have studied this material in our courses.

The book contains a rich collection of examples, exercises, and projects drawn from many fields to provide the student with a chance to solve interesting real-world problems.

The book concentrates on the principles of good software engineering and stresses program clarity through use of the structured programming methodology. We avoid the use of arcane terminology and syntax specifications in favor of teaching by example.

Among the pedagogical devices of this text are the use of complete programs and sample outputs to demonstrate the concepts being discussed; a set of objectives and an outline at the beginning of every chapter; common programming errors (CPEs) and good programming practices (GPPs) enumerated throughout each chapter and summarized at the end of each chapter; summary and terminology sections in each chapter; self-review questions and answers in each chapter; and the richest collection of exercises in any C book. An instructor's manual is available on IBM PC format disks and Macintosh format disks with all the programs in the main text and with answers to all exercises at the end of each chapter. The exercises range from simple recall questions to lengthy programming problems to major projects. Instructors requiring substantial term projects of their students will find many appropriate problems listed in the exercises for Chapters 5 through 15. We have put a great deal of effort into the exercises to enhance the value of this course for the student. The programs in the text were tested on ANSI C-compliant compilers on a Sun SPARCstation, Apple Macintosh (Think C), IBM PC (Turbo C, Turbo C++, and Borland C++), and DEC VAX/VMS (VAX C).

This text specifically follows the ANSI C standard. Many features of ANSI C will not work with pre-ANSI C versions. See the reference manuals for your particular system for more details about the language, or obtain a copy of ANSI document X3.159-1989, "*American National Standard for Information Systems—Programming Language—C*," from the American National Standards Institute, 1430 Broadway, New York, New York 10018.

## About this Book

This book is loaded with features to help the student learn.

*Objectives*

Each chapter begins with a statement of learning objectives. This tells the student what to expect and gives the student a chance, after reading the chapter, to determine if he or she has, in fact, met these objectives. It is a confidence builder and a source of positive reinforcement.

*Quotations*

The learning objectives are followed by a series of quotations. Some are humorous, some are philosophical, and some offer interesting insights. Our students have told us that they enjoy relating the quotes to the chapter material.

*Outline*

The chapter outline helps the student approach the material in top-down fashion. This, too, helps students anticipate what is to come and set a responsible pace.

*Sections*

Each chapter is organized into small sections, that address key areas. C features are presented in the context of complete, working C programs. Each program is followed by a window containing the output produced when the program is run. This enables the student to confirm that the programs run as expected. Relating outputs back to the program statements that produce the outputs is an excellent way to learn and reinforce concepts. Our programs are designed to exercise the diverse features of C. Reading the book carefully is much like entering and running these programs on a computer.

*Illustrations*

An abundance of line drawings and charts is included. The discussion of structured flowcharting, which helps students appreciate the use of control structures and structured programming, features numerous carefully drawn flowcharts. The chapter on data structures uses abundant line drawings to illustrate the creation and maintenance of important data structures such as linked lists, queues, stacks, and binary trees.

*Helpful Design Elements*

We have included four design elements to help students focus on important aspects of program development, testing and debugging, performance, and portability. We highlight scores of these in the form of *Good Programming Practices (GPPs), Common Programming Errors (CPEs), Performance Tips (PERFs),* and *Portability Tips (PORTs).*

*Good Programming Practices (GPPs)*

Good programming practices are highlighted it in the text with our GPP design element. This calls the student's attention to techniques that help produce better programs. These GPPs represent the best we have been able to glean from a combined four decades of programming experience.

*Common Programming Errors (CPEs)*

Students learning a language—especially in their first programming course—tend to make certain common errors. Focusing the students' attention on these common programming errors is an enormous help. It also helps reduce the long lines outside instructors' offices during office hours!

*Performance Enhancement Tips (PERFs)*

We find that writing clear and understandable programs is by far the most important goal for a first programming course. But students want to write the program that runs the fastest, uses the least memory, requires the smallest number of keystrokes, or dazzles in some other nifty way. Students really care about performance. They want to know what they can do to "turbo charge" their programs. So we have include *Performance Tips (PERFs)* to highlight opportunities for improving program performance.

*Portability Tips (PORTs)*

Software development has become complex and enormously expensive. Organizations that develop software must often produce versions customized to a variety of computers and operating systems. So there is a strong emphasis today on portability, i.e., on producing software that will run on many different computer systems without change. Many people tout C as the best language for developing portable software. Some people assume that if they implement an application in C, the application will automatically be portable. This is simply not the case. Achieving portability requires careful and cautious design. There are many pitfalls. The ANSI Standard C document itself lists 11 pages of potential difficulties. We include numerous Portability Tips (PORTs). We have combined our own experience in building portable software with a careful study of the ANSI standard section on portability, as well as two excellent books on portability (see references Ja89 and Ra90 at the end of Chapter 1).

*Summary*

Each of our chapters ends with a number of additional pedagogical devices. We present a detailed summary of the chapter in bullet-list fashion. This helps the students review and reinforce key concepts,.

*Terminology*

We include a Terminology section with an alphabetized list of the important terms defined in the chapter. Again, further confirmation. Then we summarize the GPPs, CPEs, PERFs, and PORTs.

*Self-Review Exercises*

Extensive Self-Review Exercises with complete answers are included for self-study. This gives the student a chance to build confidence with the material and prepare to attempt the regular exercises.

*Exercises*

Each chapter concludes with a substantial set of exercises spanning the range from simple recall of important terminology and concepts, to writing individual C statements, to writing small portions of C functions, to writing complete C functions and programs, to writing major term projects. The large number of exercises enables instructors to tailor their course to the unique needs of their audiences and to vary course assignments each semester. Instructors can use these exercises to form homework assignments, short quizzes, and major examinations. The text for the exercises is included on the IBM-PC-format and Apple-Macintosh-format disks available to instructors through their Prentice-Hall representatives.

## A Tour of the Book

Chapter 1, "Introduction," discusses what computers are, how they work, and how they are programmed. It introduces the notion of structured programming and explains

why this set of techniques has fostered a revolution in the way programs are written. The chapter gives a brief history of the development of programming languages from machine languages, to assembly languages, to high-level languages. The origin of the C programming language is discussed. The chapter includes an introduction to the C programming environment.

Chapter 2, "Introduction to C Programming," gives a concise introduction to writing C programs. A detailed treatment of decision making and arithmetic operations in C is presented. After studying this chapter, the student will understand how to write simple, but complete, C programs.

Chapter 3, "Structured Programming," is probably the most important chapter in the text, especially for the serious student of computer science. It introduces the notion of algorithms (procedures) for solving problems. It explains the importance of structured programming in producing programs that are understandable, debuggable, maintainable, and more likely to work properly on the first try. It introduces the fundamental control structures of structured programming, namely the sequence, selection (`if` and `if/else`), and repetition (`while`) structures. It explains the technique of top-down, stepwise refinement that is critical to the production of properly structured programs. It presents two program design aids, namely structured flowcharting and structured pseudocode. The methods and approaches used in Chapter 3 are applicable to structured programming in any programming language, not just C. This chapter helps the student develop good programming habits in preparation for dealing with the more substantial programming tasks in the remainder of the text.

Chapter 4, "Program Control," refines the notions of structured programming and introduces additional control structures. It examines repetition in detail, and compares the alternatives of counter-controlled loops and sentinel-controlled loops. The `for` structure is introduced as a convenient means for implementing counter-controlled loops. The `switch` selection structure and the `do/while` repetition structure are presented. The chapter concludes with a discussion of logical operators.

Chapter 5, "Functions," discusses the design and construction of program modules. C includes standard library functions, programmer-defined functions, recursion, and call-by-value capabilities. The techniques presented in Chapter 5 are essential to the production and appreciation of properly structured programs, especially the kinds of larger programs and software that system programmers and application programmers are likely to develop in real-world applications. The "divide and conquer" strategy is presented as an effective means for solving complex problems; functions enable the programmer to divide complex programs into simpler interacting components. Students enjoy the treatment of random numbers and simulation, and they appreciate the discussion of the dice game craps which makes elegant use of control structures. The extensive collection of 39 exercises at the end of the chapter includes several classical recursion problems such as the Towers of Hanoi.

Chapter 6, "Arrays," discusses the structuring of data into arrays, or groups, of related data items of the same type. The chapter presents numerous examples of both single-subscripted arrays and double-subscripted arrays. It is widely recognized that structuring data is just as important as using control structures in the development of properly structured programs. Examples in the chapter investigate various common ar-

ray manipulations, printing histograms, sorting data, passing arrays to functions, and an introduction to the field of survey data analysis. The end-of-chapter exercises include an especially large selection of interesting and challenging problems. These include improved sorting techniques, the design of an airline reservations system, an introduction to the concept of turtle graphics (made famous in the LOGO language), and the Knight's Tour and Eight Queens problems that introduce the notions of heuristic programming so widely employed in the field of artificial intelligence.

Chapter 7, "Pointers," presents one of the most powerful features of the C language. The chapter provides detailed explanations of pointer operators, call by reference, pointer expressions, pointer arithmetic, the relationship between pointers and arrays, arrays of pointers, and pointers to functions. The chapter exercises include a simulation of the classic race between the tortoise and the hare, and card shuffling and dealing algorithms. A special section entitled "Building Your Own Computer" is also included. This section explains the notion of machine language programming and proceeds with a project involving the design and implementation of a computer simulator that allows the reader to write and run machine language programs. This unique feature of the text will be especially useful to the reader who wants to understand how computers really work. Our students enjoy this project and often implement substantial enhancements.

Chapter 8, "Characters and Strings," deals with the fundamentals of processing nonnumeric data. The chapter includes an extremely detailed walkthrough of the character and string processing functions available in C's libraries. The techniques discussed here are widely used in building word processors, page layout and typesetting software, and text-processing applications. The chapter includes an interesting collection of 33 exercises that explore text-processing applications. The student will enjoy the exercises on writing limericks, writing random poetry, converting English to pig Latin, generating seven-letter words that are equivalent to a given telephone number, text justification, check protection, writing a check amount in words, generating Morse Code, metric conversions, and dunning letters. The last exercise challenges the student to use a computerized dictionary to create a crossword puzzle generator!

Chapter 9, "Formatted Input/Output," presents all the powerful formatting capabilities of `printf` and `scanf`. We discuss `printf`'s output formatting capabilities such as rounding floating point values to a given number of decimal places, aligning columns of numbers, right justification and left justification, insertion of literal information, forcing a plus sign, printing leading zeros, using exponential notation, using octal and hexadecimal numbers, and controlling field widths and precisions. We discuss all of `printf`'s escape sequences for cursor movement, printing special characters, and causing an audible alert. We examine all of `scanf`'s input formatting capabilities including inputting specific types of data and skipping specific characters in an input stream. We discuss all of `scanf`'s conversion specifiers for reading decimal, octal, hexadecimal, floating point, character, and string values. We discuss scanning inputs to match (or not match) the characters in a scan set. The chapter exercises test virtually all of C's formatted input/output capabilities.

Chapter 10, "Structures, Unions, Bit Manipulations, and Enumerations," presents a number of important features. Structures are like records in Pascal and other lan-

guages—they group data items of various types. Structures are used in Chapter 11 to form files consisting of records of information. Structures are used in conjunction with pointers and dynamic memory allocation in Chapter 12 to form dynamic data structures such as linked lists, queues, stacks, and trees. Unions enable an area of memory to be used for different types of data at different times; such sharing can reduce a program's memory requirements or secondary storage requirements. Enumerations provide a convenient means of defining useful symbolic constants; this helps make programs more self-documenting. C's powerful bit manipulation capabilities enable programmers to write programs that exercise lower-level hardware capabilities . This helps programs process bit strings, set individual bits on or off, and store information more compactly. Such capabilities, often found only in low-level assembly languages, are valued by programmers writing system software such as operating systems and networking software. A feature of the chapter is its revised, high-performance card shuffling and dealing simulation. This is an excellent opportunity for the instructor to emphasize the quality of algorithms.

Chapter 11, "File Processing," discusses the techniques used to process text files with sequential access and random access. The chapter begins with an introduction to the data hierarchy from bits, to bytes, to fields, to records, to files. Next, C's view of files and streams is presented. Sequential access files are discussed using a series of three programs that show how to open and close files, how to store data sequentially in a file, and how to read data sequentially from a file. Random access files are discussed using a series of four programs that show how to sequentially create a file for random access, how to read and write data to a file with random access, and how to read data sequentially from a randomly accessed file. The fourth random access program combines many of the techniques of accessing files both sequentially and randomly into a complete transaction processing program. Our students in our industrial seminars tell us that after studying this material on file processing, they were able to produce substantial file-processing programs that were immediately useful in their organizations.

Chapter 12, "Data Structures," discusses the techniques used to create dynamic data structures. The chapter begins with discussions of self-referential structures and dynamic memory allocation. The chapter proceeds with a discussion of how to create and maintain various dynamic data structures including linked lists, queues (or waiting lines), stacks, and trees. For each type of data structure, we present complete, working programs and show sample outputs. Chapter 12 helps the student truly master pointers. The chapter includes abundant examples using indirection and double indirection—a particularly difficult concept. One problem when working with pointers is that students have trouble visualizing the data structures and how their nodes are linked together. So we have included a large number of illustrations that show not only the actual links, but the sequence in which they are created. The binary tree example is a superb capstone for the study of pointers and dynamic data structures. This example creates a binary tree, enforces duplicate elimination, and introduces preorder, inorder, and postorder recursive tree traversals. Students have a real sense of accomplishment when they study and implement this example. They particularly appreciate seeing that the inorder traversal prints the node values in sorted order. The chapter in-

cludes a substantial collection of exercises. A highlight of the chapter is the introduction to compiling via the study of infix notation and postfix notation. The exercises walk the student through the development of an infix-to-postfix-conversion program and a postfix-expression-evaluation program. Some of our students modify the postfix evaluation algorithm to generate the machine language code a compiler would typically produce. The students place this code in a file (using the techniques of Chapter 11) and then actually run their machine language programs on the software simulators they built in the exercises of Chapter 7!

Chapter 13, "The Preprocessor," provides detailed discussions of the preprocessor directives. The chapter includes more complete information on the `#include` directive that causes a copy of a specified file to be included in place of the directive before the file is compiled, and the `#define` directive that creates symbolic constants and macros. The chapter explains conditional compilation for enabling the programmer to control the execution of preprocessor directives, and the compilation of program code. The # operator that converts its operand to a string and the ## operator that concatenates two tokens are discussed. The five predefined symbolic constants (`__LINE__`, `__FILE__`, `__DATE__`, `__TIME__`, and `__STDC__`) are presented. Finally, macro `assert` of the `assert.h` header is discussed. assert is valuable in program testing, debugging, verification, and validation.

Chapter 14, "Advanced Topics," presents several advanced topics not ordinarily covered in introductory courses. Section 14.2 shows how to redirect input to a program to come from a file, redirect output from a program to be placed in a file, redirect the output of one program to be the input of another program (piping), and append the output of a program to an existing file. Section 14.3 discusses how to develop functions that use variable-length argument lists. Section 14.4 shows how command-line arguments can be passed to function `main`, and used in a program. Section 14.5 discusses compiling programs that exist in multiple files. Section 14.6 discusses registering functions with `atexit` to be executed at program termination, and terminating program execution with function `exit`. Section 14.7 discusses the `const` and `volatile` type qualifiers. Section 14.8 shows how to specify the type of a numeric constant using the integer and floating point suffixes. Section 14.9 explains binary files and the use of temporary files. Section 14.10 shows how to use signal handling library to trap unexpected events. Section 14.11 discusses the creation and use of dynamic arrays with `calloc` and `realloc`.

We are pleased to include Chapter 15, "Object-Oriented Programming and C++." There is a revolution occurring in software engineering today. Object-oriented programming (OOP) gives us an entirely new way to view the process of designing and building software. It offers us the opportunity for software reusability instead of constantly "reinventing the wheel." With OOP, organizations are discovering that they can substantially increase productivity. Our original goal was to write a brief introduction to C++. As we wrote the chapter, however, we saw a chance to offer computer science students a major opportunity to appreciate OOP early in their education. So we decided to present a much more substantial treatment. Because of time constraints, many courses will not be able to cover this chapter in depth, if at all. Regardless, students will have the material available in the same form as we presented C—a friendly

approach with lots of learning aids and working programs. The chapter treats *both* major areas for which C++ was intended—enhancing C and supporting OOP. Among the C enhancements we discuss are single-line comments, local declarations, C++ stream-oriented input/output, inline functions, reference parameters, default arguments, and C++-style dynamic memory allocation with the `new` and `delete` operators. We then proceed with a discussion of object-oriented programming in C++. We discuss data abstraction, information hiding, classes, objects, software reusability with class libraries, the scope resolution operator, accessing class members, controlling access to data members and member functions, friend functions, constructor functions, destructor functions, inheritance, derived classes, operator overloading, function overloading, polymorphism, and virtual functions. A detailed case study on software reusability using C++ and object-oriented programming is included. The case study first converts our list processing program of Chapter 12 to C++ format. Then the program is encapsulated to form class `listclass`. We instantiate an object of that class and show that the object performs properly. Then we use inheritance to form derived classes for stacks and queues, instantiate objects of these classes, and show that these objects function properly. The reader will discover that Chapter 15 presents a solid treatment of C++ and OOP—much more than might be expected in a C textbook. We have loaded the chapter with C++ programs, execution outputs, and self-review exercises and answers, so even the student who does not have access to a C++ compiler will be able to learn a great deal. The exercises include some particularly challenging problems: using `stackclass` to create an OOP program implementing infix-to-postfix conversion and postfix evaluation algorithms, defining a string class, using OOP to develop a discrete-event queueing simulation that models the operation of a highway toll plaza, and using OOP to develop a simulation of the reader's favorite sport. The chapter includes extensive references for further study. The chapter concludes with an appendix that provides resources for the reader interested in further exploration of C++ and OOP. The appendix includes names and addresses for the Object Management Group—an industry consortium devoted to encouraging the use of OO-based techniques, publications about C++ and OOP topics, and companies that offer C++ products for the related OO-based languages Simula and Smalltalk. We hope our chapter will encourage the reader to pursue further study in C++ and object-oriented programming.

Several Appendices provide valuable reference material. In particular, we present the C syntax summary in Appendix A; a summary of all C standard library functions with explanations in Appendix B; a complete operator precedence and associativity chart in Appendix C; the set of ASCII character codes in Appendix D; and a discussion of the binary, octal, decimal, and hexadecimal number systems in Appendix E.

*Acknowledgements*

One of the great pleasures of writing a textbook is acknowledging the efforts of the many people whose names may not appear on the cover, but without whose hard work, cooperation, friendship, and understanding producing this text would have been impossible.

PJD would like to thank Professor Richard Wang of the Massachusetts Institute of Technology for his support and understanding, and for his interesting classes in C, UNIX, and SQL. PJD would also like to thank his Wellesley College friends Ms. Nancy Paz and Ms. Karin Monsler, and his MIT friends Will Martinez, Tim Nieto, Rich Wong, Mark Schaefer, Alex Hou, Goose (Brandt Casey) and last—but not least—Spike (Cliff Stephens).

HMD wants to thank his Nova University Colleagues Ed Simco, Clovis Tondo, Lois Simco, Ed Lieblein, Phil Adams, Raisa Szabo. Raul Salazar, Laurie Dringus, Pattie McCormick, and Barbara Edge.

We would like to thank our friends at the Corporation for Open Systems International (Bill Horst, David Litwack, Steve Hudson, and Linc Faurer), Informative Stages (Don Hall), Semaphore Training (Clive Lee), and Digital Equipment Corporation (Janet Hebert, Faye Napert, Betsy Mills, Jennie Connolly, Stephanie Stosur Schwartz, Barbara Couturier, John Ferreira, Gretchen Forbes, Debbie Barrett and Paul Sandore) who have made teaching this material in an industrial setting such a joy.

We are fortunate to have been able to work on this project with a talented and dedicated team of publishing professionals at Prentice Hall. Kathleen Schiaparelli did a marvelous job as production editor. Diana Penha and Jaime Zampino coordinated the complex reviewer effort on the manuscript, and were always incredibly helpful when we needed assistance—their ebullience and good cheer are sincerely appreciated.

We would like to thank Mr. James A. Cannavino, IBM Vice President and General Manager, Personal Systems, and Mr. Steve Ballmer, President of Microsoft Corporation for their friendship and for a photographic moment at Fall Comdex 1991 that we will never forget.

This book happened because of the encouragement, enthusiasm, and persistence of Marcia Horton, Vice President and Editor-in-Chief of Prentice-Hall's Computer Science and Engineering department. It is a great credit to Prentice-Hall that its top executives continue their editorial responsibilities. We have always been impressed with this, and we are grateful to be able to continue working so closely with Marcia even as her administrative responsibilities increase.

We appreciate the efforts of our reviewers

Gene Spafford (Purdue University)
Clovis Tondo (IBM Corporation and visiting professor at Nova University)
Jeffrey Esakov (University of Pennsylvania)
Tom Slezak (University of California, Lawrence Livermore National Laboratory)
Gary A. Wilson (Gary A Wilson & Associates—UNIX/C course consulting and instruction—and University of California Berkeley Extension)
Mike Kogan (IBM Corporation; chief architect of 32-bit OS/2 2.0)
Don Kostuch (IBM Corporation retired; now worldwide instructor in C, C++, and object-oriented programming)
Ed Lieblein (Nova University)
John Carroll (San Diego State University)
Alan Filipski (Arizona State University)

Greg Hidley (University of California San Diego)
Daniel Hirschberg (University of California Irvine)
Jack Tan (University of Houston)
Richard Alpert (Boston University)
Eric Bloom (Bentley College).

These people scrutinized every aspect of the text and made dozens of valuable suggestions for improving the accuracy and completeness of the presentation.

The authors would like to extend a special note of thanks to Ed Lieblein, one of the world's leading authorities on software engineering, for his extraordinary review of Chapter 15, "Object-Oriented Programming and C++." Dr. Lieblein is a friend and colleague of HMD at Nova University in Ft. Lauderdale, Florida where he is Full Professor of Computer Science. Dr. Lieblein was previously Chief Technical Officer of Tartan Laboratories, one of the leading compiler development organizations in the world. Before that, he served as Director of Computer Software and Systems in the Office of the Secretary of Defense. In that capacity, he managed the DoD Software Initiative, a special program to improve the nation's software capability for future mission critical systems. He initiated the Pentagon's STARS program for software technology and reusability, guided the Ada program to international standardization, and played an important role in establishing the Software Engineering Institute at Carnegie Mellon University. It is indeed a special privilege for us to be able to work with Dr. Lieblein at Nova University.

We would also like to extend a special note of thanks to Dr. Clovis Tondo of IBM Corporation and visiting professor at Nova University. Dr. Tondo was the head of our review team. His meticulous and thorough reviews taught us much about the subtleties of the C language and of teaching C properly. Dr. Tondo is the co-author of *The C Answer Book* which contains answers to the exercises in—and is widely used in conjunction with—*The C Programming Language*, the classic book on C by Brian Kernighan and Dennis Ritchie.

This text is based on the version of C standardized through the American National Standards Institute (ANSI) in the United States and through the International Standards Organization (ISO) worldwide. We have used extensive materials from the ANSI standard document with the express written permission of the American National Standards Institute. We sincerely appreciate the cooperation of Mary Clare Lynch—Director of Publications for ANSI—and her associates Kim Bullock and Beth Somerville for helping us obtain the necessary publication permissions. Figures 5.6, 8.1, 8.5, 8.12, 8.17, 8.20, 8.22, 8.30, 8.36, 9.1, 9.3, 9.6, 9.9, 9.16, 10.7, and 11.6, and Appendix A: C Syntax, and Appendix B: Standard Library have been condensed and adapted from *American National Standard for Information Systems—Programming Language—C, ANSI X3.159-1989*, copyright 1990 by the American National Standards Institute. Copies of this standard may be purchased from the American National Standards Institute, 11 West 42nd Street, New York, NY 10036.

Last, but certainly not least, we would like to thank Barbara and Abbey Deitel, for their love and understanding, and for their enormous efforts in helping prepare the

manuscript. They contributed endless hours of effort; they tested every program in the text, assisted in every phase of the manuscript preparation, and proofread every draft of the text through to publication. Their sharp eyes prevented innumerable errors from finding a home in the manuscript. Barbara also researched the quotes, and Abbey suggested the title for the book.

We assume complete responsibility for any remaining flaws in this text. We would greatly appreciate your comments, criticisms, corrections, and suggestions for improving the text. Please send us your suggestions for improving and adding to our list of GPPs, CPEs, PORTs, and PERFs. We will acknowledge all contributors in the next edition of the book. Please address all correspondence to our email address:

deitel@world.std.com

or write to us as follows:

Harvey M. Deitel (author)
Paul J. Deitel (author)
c/o Computer Science Editor
College Book Editorial
Prentice Hall
Englewood Cliffs, New Jersey 07632

We will respond immediately.

Harvey M. Deitel

Paul J. Deitel

# Contents