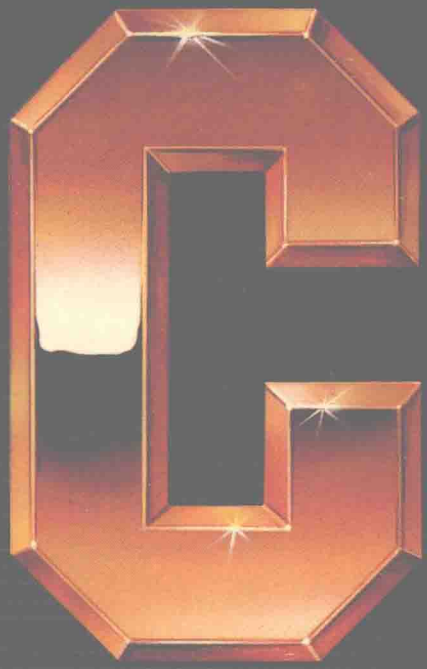


C PROGRAMMER'S LIBRARY

que™

C Programmer's Library

Jack J. Purdum
Timothy C. Leslie
Alan L. Stegemoller

Que Corporation
Indianapolis

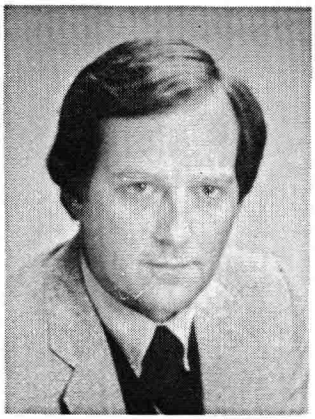
Library of Congress Catalog No.: LC 83-62490

ISBN 0-88022-048-1

88 87 86 85 84 8 7 6 5 4 3

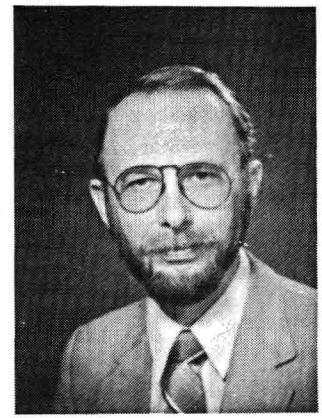
Interpretation of the printing code: the rightmost double-digit number is the year of the book's printing; the rightmost single-digit number, the number of the book's printing. For example, a printing code of 83-4 shows that the fourth printing of the book occurred in 1983.

About the Authors



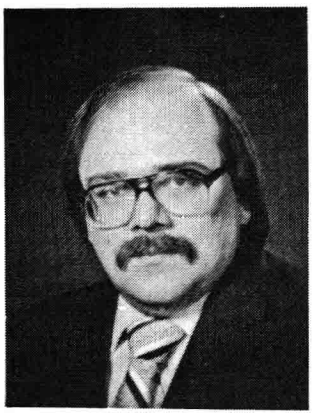
Jack J. Purdum

Dr. Purdum received his B.A. degree from Muskingum College and M.A. and Ph.D. degrees from Ohio State University. He is currently Associate Professor of Economics at Butler University, where he teaches both computer programming and economics courses. Dr. Purdum has received many teaching and research awards, including a National Science Foundation grant to study microcomputers in education. He has published a number of professional articles; a BASIC programming text; and magazine articles in *Byte, Personal Computing,* and *Interface Age.* He is also the author of the best seller, *C Programming Guide,* published by Que Corporation. Dr. Purdum is president of Ecosoft, Inc., a software house that specializes in microcomputer software.



Timothy C. Leslie

Mr. Leslie studied math and physics at Indiana University before becoming a system analyst in the U.S. Army in 1970. From 1974 to 1978 he was the Army's Chief Data Processing Branch in Berlin. Later, as a civilian, he was the system analyst for Ecocardiology with Microsonics, Inc., from 1981 to 1982. Currently, Mr. Leslie is Director of Software Development for Ecosoft, Inc.



Alan L. Stegemoller

Mr. Stegemoller received his B.S. degree in Electrical Engineering from Purdue University in 1975. From 1978 to 1983 he worked for Digilog Dynamics, Inc., and served as senior engineer in the field of hardware and software design for Medical Image Processing Systems. Currently, he is self-employed as a consulting engineer for Microsonics, Inc., a manufacturer of medical imaging hardware, and for Ecosoft, Inc. Mr. Stegemoller is coauthor of Ecosoft's Eco-C C compiler.

Editorial Director
David F. Noble, Ph.D.
Editors
Diane F. Brown, M.A.
Pamela Fullerton
Virginia D. Noble, M.L.S.
Managing Editor
Paul Mangin
Technical Editor
Chris DeVoney
Technical Consultants
Greg Dunn
James Fleming, Ph.D.

Dedication

In fond memory of my mother,
Janette B. Purdum.

J. J. P.

To my parents, Carl and Emily Leslie.

T. C. L.

To my parents, Henry and Ethel Stegemoller.

A. L. S.

Foreword

The *C Programmer's Library* is like two books in one. The first is a textbook, making suggestions and showing examples on designing and writing functions for your personal C library. This book demonstrates several ways to analyze and attack problems that confront C programmers daily or weekly. The second book contains the functions and programs. This book represents the first effort by a publisher to include extensive and highly useful C source code in book form.

In writing this book, the authors noted two facts about the C language. The core of C is portable and can be easily implemented on a variety of processors under various operating systems. This portability of C accounts for its increasing popularity. However, the C language is incomplete without an operating system. C is I/O-less and must execute under the shell of an operating system. This requires that the "standard library" be customized for each operating system.

After witnessing the testing of the code in this book on twelve different C compilers running under CP/M®, MS-DOS®, Apple® DOS, QNX™, and UNIX™ environments, I am convinced that the C standard library is not so standard, nor has the C language been fully standardized. The evident truth is that not all the functions and programs presented in this book will work on many C compilers without modifications.

To compile the functions and programs in this book, you will need a C compiler that meets or exceeds UNIX Version 7 specifications. This means that the compiler must handle `int`, `char`, `long`, `struct`, and `union` data types; `typedef`; and the `ifndef` preprocessor macro. The `float` and `double` data types are not used in this book, although the

sorting programs presented in Chapter 2 will work with `float` and `double`.

In addition, the functions presented in Chapter 5 (ISAM) and the book cataloger program in Chapter 6 make extensive use of `setjmp()` and `longjmp()` functions. These functions must be available in your C compiler for the routines in these chapters to compile and run successfully.

Possibly, a compiler may meet these specifications and yet be unable to compile the functions and programs in this book. Causes may be insufficient RAM memory, the compiler's lack of symbol or expression table space, or nonstandard functions in the "standard library"—that is, standard library functions that do not conform to Version 7 specifications.

All functions and programs were written, tested, and validated on the Eco-C compiler. All functions and programs, except those presented in Chapter 3, were tested and validated under the Portable C Compiler (`pcc`) under the UNIX operating system.

A brief word about the programs is necessary. Chapter 2 presents three sorting functions and an example disk-sorting program. Chapter 3 provides a terminal code handler and installation program. Presented in Chapter 5 is a fully functional index sequential access manager (ISAM). Chapter 6 contains a book-cataloging program that uses the ISAM functions.

The terminal handler and installation program was designed for programs running the CP/M and MS-DOS operating systems. Some CP/M and MS-DOS computers have limited disk storage capacity. Many of these machines use floppy disks. Programmers try to minimize the number of files associated with a program so that the user can easily copy the programs from one diskette to another. For these reasons the terminal installer modifies the object code; this practice is acceptable and even desirable in the CP/M and MS-DOS worlds but loathed by the UNIX programmer.

In the UNIX world, the `termcap` and `curses` libraries for terminal handling are used. The functions and programs presented in Chapter 3 are inapplicable to the UNIX environment. This is the reason the code in Chapter 3 was not validated on `pcc`.

The book cataloger in Chapter 6 required modification to compile and execute under `pcc`. The cataloger, which uses the terminal-handling routines in Chapter 3, was rewritten with the `termcap` and `curses` functions and was successfully compiled and validated. However, this modified version of the book cataloger is not presented in this book.

In Chapter 5 the ISAM functions have been heavily cast to minimize the potential conflicts between computers using different CPUs. This casting, unnecessary for 8-bit CPUs, was critical to ensure that the code would function properly with C compilers on 16-bit and 32-bit CPU computers.

Because of the differences among the CP/M, MS-DOS, and UNIX environments and corresponding C compilers, editor's notes appear at the beginning of several chapters to warn of the machine- or compiler-specific issues that must be addressed for the various compilers and operating systems.

Note that the data files produced by these programs are not guaranteed to be portable. This problem is CPU-specific. For example, some CPUs store integers in a low-byte, high-byte sequence, whereas other CPUs store integers in a high-byte, low-byte sequence.

Appendix E summarizes the modifications needed to make the functions and programs work in the UNIX environment.

Appendix F provides the results of the tests on the various CP/M and MS-DOS C compilers. This appendix also summarizes what modifications may be necessary to run the programs with other C compilers.

I wish to thank Greg Dunn, who tested the functions and programs on various C compilers running under Apple DOS, CP/M, and MS-DOS; and Jim Fleming, who validated the functions and programs under UNIX and provided helpful insights on portable programming.

For increased legibility, the functions, programs, examples, C keywords, and file names are set in a font called Digital. This font is reproduced below.

```
ABCDEFGHIJKLMNOPQRSTUVWXYZ
abcdefghijklmnopqrstuvwxyz
0123456789
!@#$%^&*()+~|
-=`\[]:;'<>",.?/{}
```

A ruler line is provided below to help you count the spaces in a program line.

```
    0    1    1    2    2    3    3    4    4    5    5
12345678901234567890123456789012345678901234567890123456789
```

Chris DeVoney

Preface

There has been a growing interest in C in the past few years; we need not repeat its strong points here. This interest is evidenced by an increasing number of introductory books on C. Such books, by necessity, must use fairly simple coding examples to teach any given concept. Our experience is that the C programmers who have mastered the first level of the language view it as a high-performance race car: they know the power is there, but they haven't yet gained the driving experience to use it to its full potential. Part of this lack of experience is that relatively little published material is commercially available that presents a view of C based on more sophisticated examples and techniques designed to teach the awesome power that C offers.

This book is designed for the C programmer who is comfortable with the general structure of C but wants to tap the second level of power that C offers. The book has three primary sections. The first is designed to teach the reader to view the C data types in a more formal manner. This is crucial for understanding complex data structures. The second section draws on material in the first section to present a general methodology for library development. The third section provides coding examples to reinforce your understanding of earlier topics that complement your C library. Although numerous routines and programs are presented throughout the text, they should not be viewed as ends in themselves.

The entire source code presented in this book is available on magnetic media. Cursor functions and the ISAM library are broken into succinct modules for easier editing and implementation. Additionally, the delete

function for ISAM (which does not appear in the book) is included on the media. This media form of the source code, which will save the programmer hours of typing and debugging, is available directly from Que for $124.95.

Many people have contributed to this text in a variety of ways. We would especially like to recognize Chris DeVoney, who served as technical editor of the project; Kim Brand for his (often comical) comments on earlier drafts of the manuscript; Jim Fleming and Greg Dunn for testing the code under a variety of C compilers and operating system environments; about three dozen people at Que, who contributed in various ways to the book; and to Bill Burton for no particular reason.

Jack Purdum
Tim Leslie
Alan Stegemoller

Indianapolis, 1984

Table of Contents

0

Laying the Groundwork

> Give a man a fish, and you feed him for a day. Teach a man to fish, and you feed him for a lifetime.
>
> — *Chinese Proverb*

Writing a program in C is fairly straightforward. Most programmers begin with an outline of the task to be performed. This outline is usually written in *pseudocode,* which is an imaginary, English-like language with some of the underlying syntax of C. (For a more complete discussion of pseudocode, see Chapter 3 of Que's *C Programming Guide.*) If properly done, the pseudocode outline should identify all the C functions needed in the program.

Usually, most of the C functions a program requires are already part of your standard C library. Although the word *standard* may suggest that all function libraries are the same, they are not (*C Programming Guide,* Chapter 1).

Some C compilers have a few dozen library functions, whereas others have more than a hundred. To make matters worse, each compiler implementer is free to assign different names to library functions that perform identical tasks.