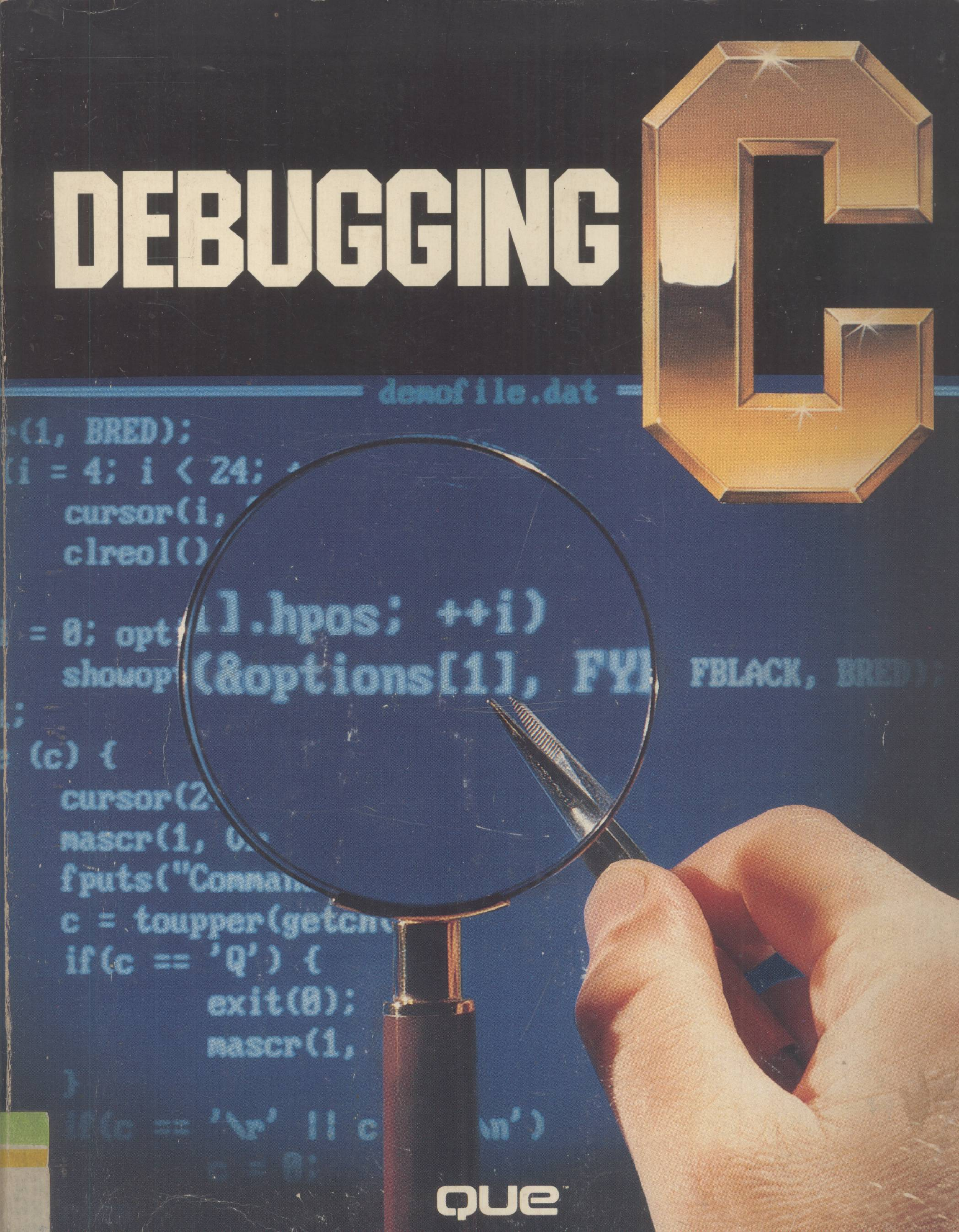
# DEBUGGING C



```
                         demofile.dat
r(1, BRED);
(i = 4; i < 24; +
    cursor(i,
    clreol()
              i].hpos; ++i)
 = 0; opt
    showopt(&options[i], FYE FBLACK, BRED);
;
e (c) {
    cursor(2
    mascr(1, 0,
    fputs("Comman
    c = toupper(getch
    if(c == 'Q') {
           exit(0);
           mascr(1,
    }
    if(c == '\r' || c      \n')
           c = 0;
```

que

# Debugging C

Robert Ward

Que™ Corporation
Indianapolis, Indiana

Library of Congress Catalog No.: LC 86-61152

ISBN 0-88022-261-1

This book is sold *as is*, without warranty of any kind, either express or implied, respecting the contents of this book, including but not limited to implied warranties for the book's quality, performance, merchantability, or fitness for any particular purpose. Neither Que Corporation nor its dealers or distributors shall be liable to the purchaser or any other person or entity with respect to any liability, loss, or damage caused or alleged to be caused directly or indirectly by this book.

90 89 88 87 86 8 7 6 5 4 3 2 1

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 87-4 shows that the fourth printing of this book occurred in 1987.

# About the Author

## Robert Ward

A digital design engineer, Robert Ward develops microprocessor-based communications equipment for an international marketer of computer accessories. He has developed programs ranging from dedicated real-time control programs for single-chip microcontrollers to small compilers for large UNIX hosts to massive vertical applications for CP/M hosts. Ward teaches computer science at McPherson College and is president of Dedicated Micro-Systems, Inc.

His quest for solutions to the challenges of programming in C on a personal computer led to his role as international coordinator of the C User's Group. Ward's interests include programming languages, computer architecture, and artificial intelligence.

*Product Director*
Chris DeVoney

*Editorial Director*
David F. Noble, Ph.D.

*Managing Editor*
Gregory Croy

*Editors*
Kathleen A. Johanningsmeier
Gail S. Burlakoff

*Technical Editor*
Paul Wilt

*Production Foreman*
Dennis Sheehan

*Production*
Joe Ramon
Mae Louise Shinault
Peter Tocco
Lynne Tone

*Composed in Megaron and Que Digital*
by Que Corporation

*Cover designed by*
Listenberger Design Associates

# Acknowledgments

There are special people who have contributed significantly to this book. I offer my heartfelt thanks to the following persons:

Chris DeVoney and the technical staff at Que, who have enlightened me about several of my programming parochialisms.

Que's editors, who enhanced the consistency and readability of the manuscripts and brought the entire work to press in a remarkably short time.

David Raanan, of AT&T Information Systems, who reviewed the discussion of sdb in Chapter 9.

The good people at C.L. Publications, who created a forum where the main thesis of this book could be tested.

Jack Purdum of EcoSoft, who offered the special encouragement and advocacy that permitted me to take this work from an idea to a published book.

# Trademark Acknowledgments

Que Corporation has made every attempt to supply trademark information about company names, products, and services mentioned in this book. Trademarks indicated below were derived from various sources. Que Corporation cannot attest to the accuracy of this information.

Ashton-Tate is a registered trademark of Ashton-Tate Company.

CodeView is a trademark of Microsoft Corporation.

CP/M is a registered trademark of Digital Research, Inc.

C-terp is a trademark of Gimpel Software.

dBASE II is a registered trademark of Ashton-Tate Company.

DDT is a trademark of Digital Research, Inc.

Eco-C88 and Eco-C Compiler are trademarks of Ecosoft, Inc.

Interactive-C is a trademark of IMPACC Associates, Inc.

IBM is a registered trademark of International Business Machines Corporation.

Lattice is a registered trademark of Lattice, Inc.

Microsoft and MS-DOS are registered trademarks of Microsoft Corporation.

PC-Lint is a trademark of Gimpel Software.

UNIX is a registered trademark of AT&T. AT&T is a registered trademark of American Telephone & Telegraph.

ZSID is a trademark of Digital Research, Inc.

Z80 is a registered trademark of Zilog, Inc.

# More Computer Knowledge from Que

| | | |
|---|---|---|
| **LOTUS SOFTWARE TITLES** | 1-2-3 for Business | $18.95 |
| | 1-2-3 Business Formula Handbook | 19.95 |
| | 1-2-3 Command Language | 19.95 |
| | 1-2-3 Financial Macros | 19.95 |
| | 1-2-3 Macro Library, 2nd Edition | 19.95 |
| | 1-2-3 Tips, Tricks, and Traps, 2nd Edition | 19.95 |
| | Using 1-2-3, 2nd Edition | 21.95 |
| | Using 1-2-3 Workbook and Disk, 2nd Edition | 29.95 |
| | Using Symphony | 23.95 |
| | Symphony: Advanced Topics | 19.95 |
| | Symphony Macros and the Command Language | 22.95 |
| | Symphony Tips, Tricks, and Traps | 21.95 |
| **DATABASE TITLES** | dBASE III Plus Applications Library | 19.95 |
| | dBASE III Plus Handbook, 2nd Edition | 19.95 |
| | dBASE III Advanced Programming | 22.95 |
| | R:base 5000 Techniques and Applications | 19.95 |
| | R:base 5000 User's Guide | 19.95 |
| | Reflex Tips, Tricks, and Traps | 19.95 |
| | Using Reflex | 19.95 |
| | Using Paradox | 19.95 |
| | Using Q & A | 19.95 |
| **APPLICATIONS SOFTWARE TITLES** | Excel Macro Library | 19.95 |
| | Multiplan Models for Business | 15.95 |
| | Using AppleWorks | 16.95 |
| | Using Dollars and Sense | 16.95 |
| | Using Enable | 19.95 |
| | Using Excel | 19.95 |
| | Using Javelin | 21.95 |
| | Using Smart | 22.95 |
| **WORD-PROCESSING TITLES** | Using DisplayWrite | 18.95 |
| | Using MicroSoft Word | 16.95 |
| | Using MultiMate | 18.95 |
| | Using the PFS Family: FILE, WRITE, GRAPH, REPORT | 14.95 |
| | Using WordPerfect, Revised Edition | 18.95 |
| | Using WordStar 2000 | 17.95 |
| **IBM TITLES** | Networking IBM PCs: A Practical Guide, 2nd Edition | 19.95 |
| | Using PC DOS | 21.95 |
| **COMPUTER SYSTEMS TITLES** | Amiga Programming Guide | 18.95 |
| | CP/M Programmer's Encyclopedia | 19.95 |
| | Managing Your Hard Disk | 19.95 |
| | MS-DOS User's Guide | 19.95 |
| | Using NetWare | 24.95 |
| **PROGRAMMING AND TECHNICAL TITLES** | Advanced C: Techniques and Applications | 21.95 |
| | C Programmer's Library | 21.95 |
| | C Programming Guide, 2nd Edition | 19.95 |
| | C Self-Study Guide | 16.95 |
| | Common C Functions | 18.95 |
| | Debugging C | 19.95 |
| | Turbo Pascal for BASIC Programmers | 16.95 |
| | Turbo Pascal Program Library | 19.95 |
| | Understanding UNIX: A Conceptual Guide | 19.95 |
| | Understanding XENIX: A Conceptual Guide | 19.95 |

Que Order Line: **1-800-428-5331**

All prices subject to change without notice.

# Table of Contents

## Introduction

## 1 Foundations

## 2 Program Testing

## 3 Localizing Compile-Time Errors

## 4 Conventional Trace Methods

## 5 Managing Trace Facilities

## 6 Why Is Debugging C Difficult?

## 7 Stabilizing Pointer Bugs

## 8 Special Trace Techniques

## 9 Source-Level Debuggers

## 10 Interpreters and Integrated Environments

## 11 Conclusion

# Introduction

## Why a Book on Debugging?

To write programs that work, you must know how to debug. It's that simple. In fact, if you produce working programs, you will spend at least half your time debugging.

To write programs that work, you must know also how to define problems (systems analysis), how to design solutions (algorithms and software engineering), and how to code in your chosen programming language (syntax and good practice). There are hundreds of books written on these subjects.

Isn't it strange that there are no books on debugging? I think so. All programmers do it, but nobody wants to talk about it and I think that's a mistake. First, you *can* improve your debugging skills. Like design skills, debugging skills evolve from critical analysis and exposure to new techniques and ideas. Second, you *must* improve your debugging skills. Good debugging skills are requisite to successful programming. To grow as a programmer and to tackle increasingly challenging assignments, your ability to debug also must grow. Because you program in C, you'll face peculiar debugging problems that demand enhanced debugging skills.

## Debugging Then and Debugging Now

When I began programming in 1969, the teaching language was Fortran IV. A well-designed program was one that got the right answer. Attitudes about program design have changed since then. Today, by means of such carefully designed teaching languages as Pascal and Logo, beginning students are introduced early to modular design. But, when I consider how I learned to debug, my introduction to programming seems rigorously structured by comparison. Except for a vague admonition to "use trace statements to find any problems," I don't remember any classroom advice about how to debug. Sadly, that situation hasn't changed much.

The popular introductory Pascal programming texts (*Oh! Pascal!*, by Doug Cooper and Michael Clancy; *Introduction to Pascal and Structured Design*, by Nell Dale and David Orshalick; and *An Introduction to Programming and Problem Solving with Pascal*, 2nd Edition, by G. Michael Schneider) include debugging tips—usually at the end of each chapter. The Clancy and Cooper sections are called "debugging and anti-bugging." I suggest that a more precise title might be "anti-bugging and anti-bugging." Except for the all-purpose admonition to include trace statements, the book sections cover defensive programming strategies, rather than debugging strategies. The authors nurture a belief that either good programmers won't have bugs or, if they do, programmers know intuitively how to find bugs.

Until I started teaching, I assumed that good debugging skills were a natural outgrowth of good design skills. Not so. A bright student may intuitively decompose a problem into beautifully coherent, cohesive, functional modules. That same student may not be able to find the most trivial syntax errors, let alone discover subtle run-time bugs. Equally bright students turn in working designs that literally defy analysis. While I don't believe that we learn debugging by studying design, I do believe that we can learn efficient debugging.

We can develop a methodological model that directs our efforts toward more productive searches. We can acquire heuristic knowledge (a kind of folk wisdom) about where to look first. We can be deliberately sensitive to the different variables and observable phenomena in different environments. We can become expert at selecting and using appropriate tools. And, through critical analysis of our attempts to find "worthy" bugs, we can learn from our own mistakes.

## Debugging = Working Programs

The simple truth is that programs seldom run right the first time. As I pointed out, debugging accounts for at least half of most development project time. Careful, disciplined design and coding won't eliminate the need for debugging; good design and coding simply make efficient debugging possible.

There is no technological relief on the horizon. The design/debug effort ratio remains fairly constant across all language classes. Neither better design nor better languages will eliminate debugging. Although debugging may change shape—we may someday find ourselves debugging specifications rather than procedures—we will need debugging as long as we create new applications.

As a teacher, I've come to believe that the greatest single difference between the person who succeeds in a programming class and the person who doesn't is that the successful programmer develops debugging skills and the unsuccessful programmer does not. While all learners commit the same types of errors in their first drafts, the individual who doesn't understand debugging remains blocked and frustrated, and the individual who does understand debugging finds the errors and goes on to experiment with new and more powerful techniques.

You may discover that your growth as a programmer has been restricted by your debugging skills. Think of a time when you were frustrated by a new operating system or language. Was part of your difficulty caused by your inability to find bugs in the new environment? To restate the question, if I could guarantee that you would find a new bug after only two seconds of searching, would you be willing to tackle almost any project in any environment? Successful debugging supports exploration and refines understanding. The more adept you are at finding bugs in a given environment, the more readily you can master that environment.

## Debugging C Is Demanding

You can't rely on debugging techniques borrowed from other programming languages when you program in C. C programmers willingly discard the protection provided by other high-level languages. Too often, newcomers underestimate the impact of this change on the debugging environment. Novices assume that because both C and Pascal are small and well-structured languages, debugging C is similar to debugging Pascal. Nothing could be farther from the truth. Some of the special problems associated