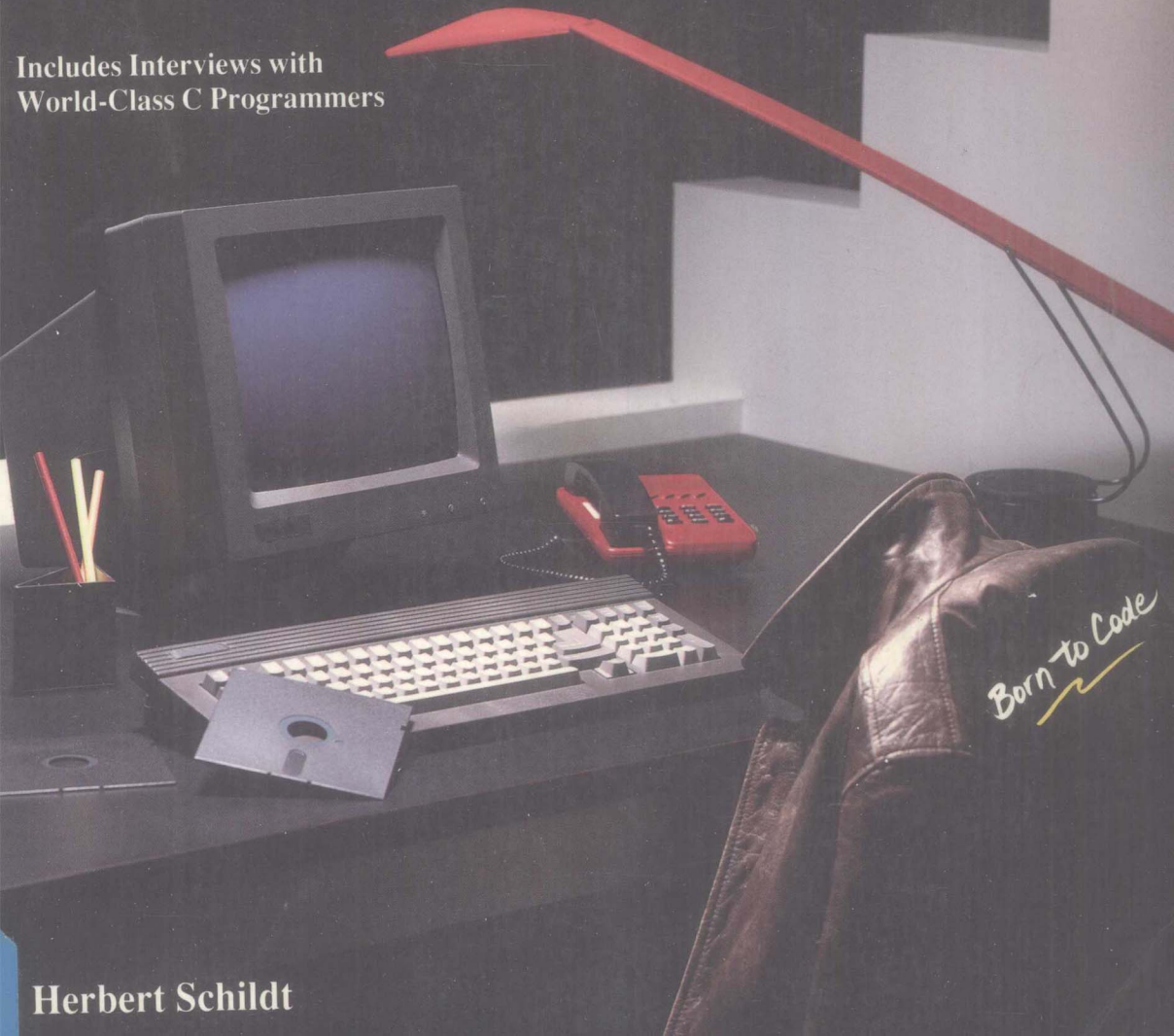
Osborne McGraw-Hill

Born to Code in C

Includes Interviews with
World-Class C Programmers



Herbert Schildt

Born to Code in C

Herbert Schildt

Osborne **McGraw-Hill**

Berkeley New York St. Louis San Francisco
Auckland Bogotá Hamburg London Madrid
Mexico City Milan Montreal New Delhi Panama City
Paris São Paulo Singapore Sydney
Tokyo Toronto

Osborne **McGraw-Hill**
2600 Tenth Street
Berkeley, California 94710
U.S.A.

For information on translations and book distributors outside of the U.S.A., please write to Osborne **McGraw-Hill** at the above address.

A complete list of trademarks appears on page 519.

Born to Code in C

234567890 DOC 89

ISBN 0-07-881468-5

This book is dedicated to Dennis Ritchie, the creator of the C language, and to programmers everywhere who were born to code in C.

PREFACE

This book is about C and C programmers. In its nine chapters you will find examples of C applied to a wide variety of applications. You will also find profiles of 14 of the world's finest C programmers. They describe how they discovered C, what their own personal design philosophies are, and how they approach programming.

These programmers were chosen because they were either instrumental to the advancement of the C language or because they have used C to create successful programs. I have attempted to present a representative cross-section of programmers. (Of course, it was not possible to include every worthy C programmer.)

In the process of writing the programmers' profiles, I noticed two traits common to all: drive and enthusiasm. Without fail, these are people who like their work. In fact, most would probably not refer to programming as work, but as a way of life. The creation of the programmer profiles was one of the most enjoyable writing tasks I have performed and I thank all who participated for their time and effort.

About This Book

The topic of each chapter was chosen because it met at least one of the following conditions: useful, interesting, unique, or fun. Chapter 1, the Little C Interpreter, is probably my favorite for one reason: it's just plain fun! Although language interpretation has several practical uses, I developed the Little C interpreter simply for the joy of doing it.

Chapter 2 explores icon-based interfaces. It includes a complete icon editor as well as an icon-based DOS interface as an example.

If you work (or play) in a DOS environment, you will find Chapters 3 and 4 particularly interesting and useful. Chapter 3 is about supercharging TSRs. You'll learn how to construct a memory-resident program that

can perform sophisticated screen and disk I/O—and develop a full-featured TSR program in the process. Chapter 4 was very exciting to work on. It is a complete multitasking subsystem for DOS. It intercepts the computer's real-time clock interrupt and uses it to multitask individual parts of a C program. This is no toy—it really works!

Chapters 5 and 6 develop two useful subsystems: a screen editor and a database manager. These subsystems can be integrated into application programs as user-amenities.

Chapter 7 examines the creation of custom character fonts. It includes a complete font editor for creating characters and a custom character display subsystem. Custom character fonts can give your programs a unique appearance.

Chapter 8 looks at animation. It develops an animation editor with which you can define objects and their motion. It also develops an animation subsystem that can be included in your application programs. Mouse interfacing is also discussed.

Chapter 9 concludes the book with a look at controlling Epson (and compatible) printers. The Epson printers have many capabilities that are often overlooked. The chapter develops both text and graphics functions that take advantage of many of the Epson's more advanced features.

Who is this book for?

This book is for any C programmer. If you are just beginning, then you can use the programs "as is" until you become proficient. If you are an experienced programmer, you can use the functions and routines presented as starting points for your own applications.

Diskette Offer

There are many useful and interesting programs contained in this book. If you're like me, you probably would like to use them, but hate typing them into the computer. When I key in routines from a book it always seems that I type something wrong and spend hours trying to get the program to work. For this reason, I am offering the source code on diskette with all the functions and programs contained in this book for $24.95. Just fill in the order blank on the next page and mail it, along with your payment, to the address shown. Or, if you are in a hurry, just call (217) 586-4021 (the number of my consulting office) and place your order by telephone. (Visa and Mastercard accepted.)

HS

Mahomet, Illinois
March, 1989

Please send me _____ copies, at $24.95 each, of the programs in *Born to Code in C*. Foreign orders, please add $5 for shipping and handling.

Name_____

Address_____

City _____ State _____ Zip _____

Telephone_____

Diskette size (check one): 5 1/4" _____ 3 1/2"_____

Method of payment: check Visa MC

Credit card number: _____

Expiration date:_____

Signature:_____

Send to:_____

Herbert Schildt
RR 1, Box 130
Mahomet, Il 61853

or phone: (217) 586-4021

Osborne/McGraw-Hill assumes NO responsibility for this offer. This is solely an offer of Herbert Schildt and not of Osborne/McGraw-Hill.

The manuscript for this book was prepared and submitted to
Osborne/McGraw-Hill in electronic form.
The acquisitions editor for this project was Jeffrey Pepper,
the technical reviewer was Tom Green,
and the project editor was Nancy Beckus.

Text design by Judy Wohlfrom, using Zaph for text body and for display.

Cover art by Stephen Black Design, Inc.
Color separation by Colour Image;
cover supplier, Phoenix Color Corporation.
Screens produced with InSet, from InSet Systems, Inc.
Book printed and bound by R.R. Donnelley & Sons Company,
Crawfordsville, Indiana.

CONTENTS

CHAPTER ONE

A C Interpreter

Language interpreters are fun! And what could be more fun for a C programmer than a C interpreter?

To begin this book I wanted a topic that would be of interest to virtually all C programmers. I also wanted the topic to be fresh, exciting, and useful. After rejecting many ideas, I finally decided upon the creation of the Little C interpreter. Here's why.

As valuable and important as compilers are, the creation of a compiler can be a difficult and lengthy process. In fact, just the creation of a compiler's run-time library is a large task in itself. By contrast, the creation of a language interpreter is an easier and more manageable task. Also, if correctly designed, the operation of an interpreter can be easier to understand than that of a comparable compiler. Beyond ease of development, language interpreters offer an interesting feature not found in compilers, an engine that actually executes the program. Remember, a *compiler* only translates your program's source code into a form that the computer can execute. However, an *interpreter* actually executes the program. It is this distinction that makes interpreters interesting.

PROFILE

Ralph Ryan
Project Manager of Microsoft C Version 3.0

Ralph Ryan has been very active in the C programming community. He was project manager for Microsoft's C, version 3.0, and then manager of Compiler Technology at Microsoft when version 4.0 was prepared. He coauthored (with Tom Plum) a C compiler test suite—Plum Hall Validation Suite—that became the industry standard. He has served on the ANSI standardization committee and is the author of a book about Microsoft's LAN Manager. These accomplishments are all the more impressive given that Ralph was first introduced to C only ten years ago, in 1979, when he was building general-purpose process-control systems. As Ralph puts it, "My prior professional experience had all been in FORTRAN or Assembler. But after learning C, I knew that there could be no going back."

As project manager of one of the most popular C compilers, Ralph is particularly proud of one innovation added by the Microsoft team to the 8086 (family) based C compilers: "The most important extensions we (Microsoft) added to C compilers for the 8086 family of processors were the **near** and **far** modifiers. These extensions let programmers take advantage of the Intel architecture to create high performance programs. No longer did a program have to be compiled for just one memory model. Instead, the programmer could control on a function-by-function and variable-by-variable basis how each element in the program would be treated."

Ralph has this to say about his personal design philosophy: "I resist the urge to plunge into coding until I have all of the details sketched out, usually into fairly low level pseudo-code. I gradually refine this into real code and add several debugging assertions as I go. In this way, I hope to minimize debugging and gain a more coherent design in the process."

Ralph gives this advice to C programmers: "C is a powerful tool. But, it is just a tool. Clarity and organization are the keys to creating great programs. Become a great programmer and you will get the most out of C."

Ralph left Microsoft after working there for over six years—a long time in the programming business. He is currently writing books about programming. He lives in Bellevue, Washington and when he isn't writing or programming, he enjoys playing in a local rock and roll band.

If you are like most C programmers, you use C not only for its power and flexibility but also because the language itself represents an almost intangible, formal beauty that can be appreciated for its own sake. In fact, C is often referred to as "elegant" because of its consistency and purity. Much has been written about the C language from the "outside looking in," but seldom has it been explored from the "inside." Therefore, what better way to begin this book than to create a C program that interprets a subset of the C language?

In the course of this chapter an interpreter is developed that can execute a subset of the ANSI C language. Not only is the interpreter functional, but it is also well-designed—you can easily enhance it, extend it, and even add features not found in ANSI C. If you haven't thought about how C really works, you will be pleasantly surprised to see how straightforward it is. The C language is one of the most theoretically consistent computer languages ever developed. By the time you finish this chapter, you will not only have a C interpreter that you can use and enlarge but you will also have gained considerable insight into the structure of the C language itself. If you're like me, you'll find the C interpreter presented here just plain fun to play with!

Note: The source code to the C interpreter presented in this chapter is fairly long, but don't be intimidated by it. If you read through the discussion, you will have no trouble understanding it and following its execution.

THE PRACTICAL IMPORTANCE OF INTERPRETERS

Although the Little C interpreter is interesting in and of itself, language interpreters do have some practical importance in computing.

As you probably know, C is generally a *compiled language.* The main reason for this is that C is a language used to produce commercially salable programs. Compiled code is desirable for commercial software products because it protects the privacy of the source, prevents the user from changing the source code, and allows the programs to make the most efficient use of the host computer, to name a few reasons. Frankly, compilers will always dominate commercial software development, as they should; however, any computer language can be compiled or interpreted. In fact, in recent years a few C interpreters have appeared on the market.

There are two traditional reasons for using a C interpreter. First, beginners to C sometimes find the (potentially) highly interactive environment of an interpreter preferable to a compiler (although this is changing since the introduction of C-integrated programming environments, such as Borland's Turbo C and Microsoft's QuickC).

The second reason is debugging. The advantage that an interpreter has over a compiler in debugging is that the contents of all variables can be known and changed at any time. Also, a C interpreter can provide trace facilities that are difficult to equal in a compiled environment.

Another practical use you might have for an interpreter is as the basis for a database *query language.* Virtually all database query languages are interpreted because of the nature of the task. Although using C as a query language would not be appropriate in most circumstances, many of the principles you learn here will be.

There is another reason that language interpreters are interesting: they are easy to modify, alter, or enhance. This means that if you want to create, experiment, and control your own language, it is easier to do so with an interpreter than a compiler. Interpreters make great language prototyping environments because you can change the way the language works and see the effects very quickly.

Interpreters are (relatively) easy to create, easy to modify, easy to understand, and, perhaps most important, fun to play with. For example, you can rework the interpreter presented in this chapter to execute your program backward—that is, executing from the closing brace of **main( )** and terminating when the opening brace is encountered. (I don't know why anyone would want to do this, but try getting a compiler to execute your code backward!) Or, you can add a special feature to C that you (and perhaps only you) have always wanted. The point is that while compilers absolutely make more sense when doing commercial software development, interpreters let you really have fun with the C language. It is in this spirit that this chapter was developed. I hope you will enjoy reading it as much as I enjoyed writing it!

THE LITTLE C SPECIFICATIONS

Despite the fact that ANSI C has only 32 *keywords* (built-in commands), C is a very rich and powerful language. It would take far more than a single chapter to fully describe and implement an interpreter for the entire C

language. Instead, the Little C interpreter understands a fairly narrow subset of the language. However, this particular subset includes many of C's most important aspects. What to include in the subset was decided mostly by whether it fit one (or both) of these two criteria:

1. Is the feature fundamentally inseparable from the C language?
2. Is the feature necessary to demonstrate an important aspect of the language?

For example, features such as recursive functions and global and local variables meet both criteria. The Little C interpreter supports all three loop constructs (not because of the first criterion, but because of the second criterion). However, the **switch** statement is not implemented because it is neither necessary (nice, but not necessary) nor does it demonstrate anything that the **if** statement (which is implemented) does not. (Implementation of **switch** is left to you for entertainment!)

For these reasons, I implemented the following features in the Little C interpreter:

- Parameterized functions with local variables
- Recursion
- The **if** statement
- The **do-while**, **while**, and **for** loops
- Integer and character variables
- Global variables
- Integer and character constants
- String constants (limited implementation)
- The **return** statement, both with and without a value
- A limited number of standard library functions
- These operators: +, −, *, /, %, <, >, <=, >=, ==, !=, unary −, and unary +
- Functions returning integers
- Comments