The C Toolbox

William James Hunt

The C Toolbox

William James Hunt

Addison-Wesley Publishing Company, Inc.
Reading, Massachusetts Menlo Park, California
Don Mills, Ontario Wokingham, England Amsterdam
Sydney Singapore Tokyo Mexico City Bogotá
Santiago San Juan

Library of Congress Cataloging in Publication Data

Hunt, William James.
The C toolbox.

Bibliography: p.
Includes index.
1. C (Computer program language) I. Title.
QA76.73.C15H85 1985 001.64'24 85-6171
ISBN 0-201-11111-X

ABCDEFGHIJ-HA-898765
First printing, July 1985

Cover design by Marshall Henrichs
Text design by Judith Ashkenaz
Set in 10 point Caledonia by Pine Tree Composition, Inc., Lewiston, Maine

Acknowledgments

Several years ago I read *Software Tools* by Kernighan and Plauger. It helped me to understand my years of programming experience. This book is inspired by *Software Tools* and uses a similar approach to systems programming on personal computers.

In the 11 years that I have been a professional programmer, I have learned from many co-workers. Reading programs they had written was an essential part of my education.

My wife, Lesley, has been invaluable in many ways. Her support encouraged me to start writing a book (and to finish it). Her work in serving as editor and guinea pig for the roughest of drafts was vital too. Finally, she volunteered to learn C to be a better guinea pig. Greater love hath no spouse than to learn another programming language.

Introduction

Anyone who has used a personal computer (PC) with good software such as Lotus's 1-2-3 or Microsoft's Flight Simulator knows that PCs have great potential. However, the best efforts of casual programmers working in BASIC are often very disappointing. There is a wide gap between what average programmers can achieve and what they see professional programmers achieve. This book helps bridge that gap. It presents examples of complete, useful programs and discusses how they were developed.

The programs are mostly utilities or tools programs that make using and programming a PC easier. I chose these programs to illustrate programming techniques and algorithms that are often mysterious to the casual BASIC programmer. The programs also provide a concrete basis for discussing program design, coding, and testing.

The book is also intended to be a showcase for the C language. If your experience is limited to BASIC, learning C (or another good high-level language like Pascal) is a necessary first step for producing high-quality programs. While the book is not a tutorial or primer on C, it provides concrete examples of its use and is a good tool for learning C.

Programs

☐ To learn about programming (and C), you need to write programs. This book provides a good starting point. Each program is simple so that its basic structure is obvious and is accompanied by a full description of what the program does and how it does it. At the same time, the programs are designed to be easy to enhance for better performance. Possible enhancements are discussed for each program, with many modifications that require only 10 to 50 lines of C code to produce useful improvements. This is much more rewarding than having to start from scratch.

To make programs easier to understand and modify, the book uses a subset of C's features. C programs are often written in a cryptic style that makes heavy

use of features unique to C. Using these features usually produces only a modest improvement in performance but a heavy penalty in program clarity. This view may be heresy to some C programmers but will prove eminently practical to many others, especially those coming to C from another language. Optimization by using C's special features is discussed in Chapter 9.

The material presented is substantial. If you do not understand a program fully at first reading, that is quite natural. Read first for general understanding, then focus on details as you need them.

The programs are designed to run on the IBM PC under PC-DOS. The nature of the programs requires that some specific environment be selected, and the IBM PC has important advantages:

1. It is widely used and is becoming a standard.
2. It is more accessible than a large computer to many readers.
3. It has a good architecture, satisfactory speed, and can use lots of memory.
4. It has several good C compilers available (see appendix B).

Only a few changes are needed to make the programs run on other computers under the MS-DOS operating system. More changes are needed to make them run under other operating systems, such as CP/M-80, CP/M-86, or UNIX. These changes are largely confined to Chapter 7.

Audience

☐ Readers should be familiar with some programming language and with using a PC, especially using a text editor and DOS commands such as DIR and COPY. It is not essential that you have previous experience with C. Programmers with exposure to Pascal, PL/1, or Modula-2 should have little difficulty. Readers with only BASIC experience are likely to find some features of C puzzling; the examples in Chapters 1 and 2 should help. My style of creating code, by stressing clear, readable code over speed, should also help the transition to C.

Available Disk

☐ All the programs in the book are available on a disk, ready for editing and compilation. Since manually copying several hundred lines of code is tedious and can result in errors, the disk is the convenient, safe way to work with these programs. Just insert the disk and edit one of the programs.

The programs were tested with many of the available compilers. Operating instructions for compiling and using the programs with various compilers are included on the disk.

The disk includes all the programs in this book plus additional programs. Library functions for scanning a single directory and for scanning a hierarchy of directories are included. These were not included in the book, to keep the book at a manageable size, but it was decided to keep all useful programs on the disk. To acquire the disk, send a check or money order for $20.00 to William James Hunt, Toolbox Disk, P.O. Box 271965, Concord, CA 94527. The last page of this book has an order form.

What You Need

☐ You should get a copy of *The C Programming Language* by Brian W. Kernighan and Dennis M. Ritchie (Englewood Cliffs, N.J.: Prentice-Hall, 1978). While the book is not a tutorial or a primer, it is the standard reference on C. You may also want a primer on C; see Appendix D for suggestions. If you are new to C, you should read one of these primers as you go through this book.

You will need free access to an IBM PC or some other computer running the MS-DOS or PC-DOS operating system (Version 2.0 or later). You will neither find programming enjoyable nor the advice in this book useful if you have limited and infrequent access to a PC.

Your computer should have at least 192 K bytes of RAM memory. It should also have either two 320 K/360 K byte floppy disk drives or one floppy and one hard disk. Appendix A gives some hints to make program development faster.

You will need an editor program to enter new programs and to modify existing ones. The EDLIN editor provided with MS-DOS is not recommended for regular use (see Appendix A for alternatives). You also need a good C compiler; Appendix B discusses specific changes required for a number of C compilers.

A few functions in the book are written in assembler language. The program diskette contains these functions in source file format as well as in object module form for several popular compilers. If you have one of these compilers, you will need an assembler only if you modify these source files.

You will need a linker program to build a complete program from separate object files. PC-DOS includes a linker that can be used with several compilers. (Other compiler products include a suitable linker program.)

Compatibility

☐ All the programs presented have been tested on the IBM PC, XT, and AT models. The programs have not been tested with the PC jr. but should work if enough memory is present.

Some hardware-specific functions in Chapter 7 may require revisions for new PC hardware. For example, the SCREEN module for fast output to the display

screen works for the monochrome and color graphics adapters. New adapters may require changes to refresh buffer addresses and status port numbers.

New releases of C compilers may differ in memory layout or in the names of segments and groups. Calling conventions for functions may also change.

Outline of the Book

☐ Chapters 1 and 2 get you started reading and writing C. While they provide a brief introduction in comparison to that of a primer, they illustrate some features and concepts that are important in understanding the rest of the book. Chapter 1 gives examples of short C programs and the steps needed to create and compile them. It shows those parts of C that will be relatively familiar to you. This gives you confidence that C will be easy to understand.

Programming languages are never perfect; Chapter 2 discusses the way we use C to minimize bugs and portability problems. File I/O and bit operations in C are also covered in this chapter. We also develop some tools and start an object library.

Chapter 3 presents the VIEW program. It allows us to browse through text files composed of ASCII characters. VIEW is useful for looking at C source files as well as text files produced by word processors like WordStar. Techniques for simple file input and for interactive keyboard and display usage are demonstrated. Since Chapter 3 is the first chapter to describe a sizeable program, it goes through the program development process in detail. The following chapters do not repeat these basic points. Testing the program on a module-by-module basis is emphasized here.

Chapter 4 develops another tool for examining data files. This FILEDUMP program displays a file's contents in hexadecimal and ASCII formats much like the dump format of the PC-DOS debug program. This provides a way to determine the format of any file without any prior knowledge of its content. Since this program is closely related to the VIEW program of Chapter 3, Chapter 4 shows how to make use of an existing program in developing a new, but related, one. This chapter discusses more aspects of the program development process.

Chapter 5 builds several tools based on sorting algorithms. Two well-known algorithms—the insertion sort and quicksort—are shown and then made into library functions that are usable on any kind of data. These library functions are then used in a simple program to sort lines of ASCII text. Then the technique of merging is used to handle files too large to fit into RAM memory. The initial MERGE program is then generalized so that the type of records to be sorted and the location and type of the keys for sorting are specified when the program is executed.

Chapter 6 builds a BTREE module for indexed access to data files. This module is rather long but provides features such as multiple indexes to a data file,

duplicate keys, and variable-length keys. A sample application maintains an index of correspondence documents by name of addressee, date sent, and subject.

Chapter 7 presents the toolkit used in preceding chapters. While C is adequate for almost all our requirements, a few things require assembler language functions. Other modules specific to MS-DOS or the IBM PC are presented in this chapter. Single-key input, screen output, and DOS and BIOS access are among the modules included here.

Chapter 8 presents a terminal emulator program. The special characteristics of a real-time program are discussed—for example, unpredictable input from several sources with data being lost if the program does not keep up with all input. The techniques for handling such problems—polling input status, buffering data to relax timing requirements, and using interrupt-driven input handling—are demonstrated.

Chapter 9 covers a few loose ends—for example, using C's unique features to optimize execution speed and handling critical errors. It also summarizes the design philosophy of the previous chapters.

Appendix A discusses compiling and executing the programs we present. Appendix B lists the C compilers examined and specific changes required for each compiler. Appendix C explains memory models and their relation to the 8088 architecture. A short bibliography in Appendix D provides a starting point for further reading.

Themes

☐ Several themes occur throughout the book. They are listed here to help you understand the book's message.

Create Order out of Chaos

The book presents finished, working programs that, I hope, do not contain bugs. It often looks as though such programs sprang forth from my brain in a complete and correct form. That is almost never the case. The process usually involves some dead ends, some backtracking, lots of bugs, and inelegant first tries.

Each chapter presents a topic in orderly way to make it easy to understand. Do not conclude that program development is a perfectly orderly process with uniform progress through each stage, however. Every program in this book was revised several times to make it simpler and more effective.

In spite of these limitations, you should always set goals at each step. Never start writing a program until you define its function. After you complete the program, you may revise the functional specification and rewrite the program. The cardinal sin is to write a program and then figure out what function it performs.

Keep Up Your Morale

Writing computer programs can be very hard on your ego and your morale. You will make errors in design, get the syntax of C statements wrong, and produce lots of very puzzling bugs in your programs. These problems give abundant and painful testimony to your lack of perfection before you finally get a working program. Some people claim to produce working programs without any such problems, but most just have memory lapses.

You can make the programming process much more pleasant if you break the problem into small pieces. Any large project can seem impossible if you do not break it into manageable steps. Write and test programs in small modules so you get some measure of success at frequent intervals. Simplify the design of a large program, and do a smaller prototype first.

Use an Experimental Approach

Reading reference manuals and computer documentation is hard, frustrating work. The information they contain is often ambiguous and incomplete. This is a fact of life for beginners and old-timers alike. The way to cope with bad documentation is to perform an experiment. Writing a short C program to see how a feature actually works is more productive and less frustrating than guessing what a cryptic description in a manual really means.

Tackle such problems one at a time. It is much easier to understand how a feature of C or the operating system works if you try it out in a controlled environment. Writing a sizeable program filled with unverified assumptions about such features makes testing unnecessarily difficult.

Learn by Doing

Do not waste too much time at first studying C. Learn a little and plunge in. Start by reading a C program for general understanding of what it does. Then think of a small improvement you would like to make. You should be able to find enhancements that require adding fewer than ten lines to the program. Concentrate on understanding how your change will affect the program—what has to be changed and how you can accomplish the change using C.

Learn C as you need it. Develop a small working vocabulary of C features, and expand it as needed. You can treat much of C as material to be looked up when you need it, just as you would unfamiliar words.

Copy Techniques from Existing Programs

When you want to do something new, look through this book for examples that do something similar to what you want. After you copy a technique several times, you learn it without effort.

Contents

1
A Quick Tour through C

This chapter shows what C programs look like. The short programs shown do not perform useful functions, but they illustrate C's basic features. The book is not a tutorial on C, but this chapter provides a foundation for later chapters. Vocabulary and concepts needed in the rest of the book are also discussed here. More advanced features of C are discussed in later chapters.

There is a lot of similarity between different high-level computer languages. If you are familiar with BASIC, COBOL, PL/1, Pascal, or FORTRAN, you should be able to recognize the purpose of the features introduced and to relate them to the corresponding features in the language you know. If you have no previous experience with C, one of the primers on C listed in Appendix D might make learning the language easier. If you are familiar with C, this chapter can serve as a review. You may also be familiar with the terms and concepts introduced, but the chapter explains how the book uses them.

1.1 HELLO Program: The Structure of C

☐ The first program is very simple; it displays

```
hello, world
```

on the screen. (The program is not an original composition. I took it from Kernighan and Ritchie [1978], hereafter referred to as K & R.) It serves to illustrate the basic structure of a C program and how to convert it into an executable program.

Figure 1.1 lists the program. Line 1 tells the compiler to use the contents of a file named stdio.h as additional input to the compilation. The file stdio.h is normally provided with the compiler product. For now, we consider it as magic we recite in each C program.

Figure 1.1 hello.c

```
1    #include "stdio.h"
2
3    main()
4     {
5       printf("hello,") ;
6       printf(" world") ;
7     }
```

Lines 3 to 7 define a *function* named main. Line 3 establishes the function's name, and lines 4 to 7 describe what it does. This general format is followed for all C functions:

```
name(. . .)
  . . .
  {
  what it does
  }
```

As the ellipses indicate, there may be more text that we have not illustrated or described. C programs are composed of function definitions. Each definition describes what happens when we *call* that function. When we execute the HELLO program, we call the main function. In turn, it may call other functions.

The what-it-does part of a function consists of one or more *statements* (if the function does not do anything, there would be no statements). Lines 5 and 6 in the HELLO program are such statements. Line 5 calls a function named printf. printf displays the message on the screen. When it finishes, it *returns* to the point where it was called. When we call printf, we specify what it is to display. In line 6 we call printf again with a different message. What printf does depends on the information we provide it. This information is called *parameters*, or *arguments*.

The printf function is supplied in the standard C library; we do not have to write it. printf's use should be documented in your C compiler manual. With this documentation, we can use printf without seeing how it is implemented.

The statements illustrated in Figure 1.1 have the form

```
function-name(. . .) ;
```

Later programs show other forms of statements and relate the statements shown here to a more general form.

The arguments we use in lines 5 and 6 are *character string constants*. The format of such a character string constant is

"printable characters"

Compiling and Executing the HELLO Program

Figures 1.2a–d illustrate the process of preparing the HELLO program on an IBM PC using the Lattice C compiler. While the details depend on the compiler product you use, similar steps for editing, compiling, linking, and executing the programs are followed.

First we type in the C program using a text editor (or a word processor). When we finish typing the program, we store the text entered in a file. Figure 1.2a shows the editing process using the EDLIN editor provided with PC-DOS. EDLIN is used in the illustration since everyone using PC-DOS has it. Any text editor can be used as long as it produces files of ASCII characters without any special formatting information. We refer to the file created as a *source file* since it is the source for the compilation step.

The compiler reads the source file and translates the C program into instructions and data the IBM PC can execute. Figure 1.2b shows the compile process. Typing lc runs the lc.bat batch file that executes two programs, lc1 and lc2. Some C compilers consist of four separate programs, but the overall result in any case is to produce executable instructions and data at the end. By convention, C source files are normally named with a file extension of .c. Some compilers require this naming scheme. The file produced by the compiler is called an *object file*.

The compile step translated the source file into executable form, but there is another step before the program can be executed. The program that we wrote is not complete because it uses the printf library function. The linking step combines our object file with any necessary functions to produce a complete program that is ready to execute. The program is stored in a *run file* named hello.exe (if your compiler package includes a special linker, it may name the run file differently.)

Figure 1.2c shows the linking step. A special object file named cs.obj, supplied with the Lattice C compiler, sets up the environment expected by C functions; it is always the first object file specified in the link command. The last name on the line specifies that the lcs.lib library is to be searched for any library function that hello.obj requires.

Now we are ready to execute the program. Type the name of the program's run file to execute it. Figure 1.2d shows execution of the HELLO program. Note that we need not type the complete file name—the .exe extension is assumed. When we execute hello.exe, we are calling the main function. C expects to find a function named main in every program; when we named our function main, we were saying to the C compiler, "When you execute the program, begin here."

Figure 1.2 hello.fig

```
 1
 2                Figure 1.2 - Creating and Executing Hello
 3
 4   Figure 1.2a - Using an Editor
 5
 6   D>edlin hello.c
 7   New file
 8   *i
 9        1:*#include "stdio.h"
10        2:*
11        3:*main()
12        4:* {
13        5:*   printf("hello,") ;
14        6:*   printf(" world") ;
15        7:* }
16        8:*^C
17   *e
18
19   D>
20
21   Figure 1.2b - Compiling the Program
22
23   D>lc hello
24
25   D>LC1 hello
26   Lattice C Compiler (Phase 1) V2.00
27   Copyright (C) 1982 by Lattice, Inc.
28
29   D>LC2 hello
30   Lattice C Compiler (Phase 1) V2.00
31   Copyright (C) 1982 by Lattice, Inc.
32   Module size P=0017 D=000E
33
34   Figure 1.2c - Linking the Program
35
36   D>link cs hello , hello , nul , lcs
37
38   IBM Personal Computer Linker
39   Version 2.00 (C) Copyright IBM Corp 1981, 1982, 1983
40
41   D>
42
43   Figure 1.2d - Executing the Program
44
45   D>hello
46   hello, world
47   D>
```