Brady

C DEVELOPMENT TOOLS FOR THE IBM PC

Access to Advanced Features
For Serious Programmers

AL STEVENS

C Development Tools for the IBM PC

Al Stevens

A Brady Book
Published by Prentice Hall Press
New York, New York 10023

C Development Tools for the IBM PC

A Brady Book
Published by Prentice Hall Press
A Division of Simon & Schuster, Inc.
Gulf & Western Building
1 Gulf & Western Plaza
New York, New York 10023

PRENTICE HALL PRESS is a trademark of Simon & Schuster, Inc.

Manufactured in the United States of America

1 2 3 4 5 6 7 8 9 10

Library of Congress Cataloging-in-Publication Data
Stevens, Al, 1940–
C development tools for the IBM PC.
Includes index.
1. C (Computer program language) 2. IBM Personal Computer—Programming. I. Title.
QA76.73.C15S73 1986 005.265 85-16660

ISBN 0-89303-612-9

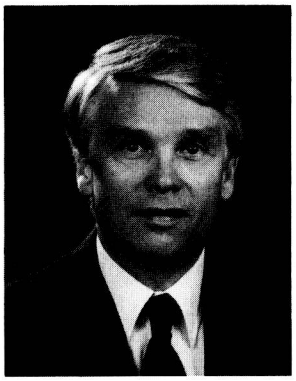
Limits of Liability and Disclaimer of Warranty

The author(s) and publisher of this book have used their best efforts in preparing this book and the programs contained in it. These efforts include the development, research, and testing of the theories and programs to determine their effectiveness. The author(s) and publisher make no warranty of any kind, expressed or implied, with regard to these programs or the documentation contained in this book. The author(s) and publisher shall not be liable in any event for incidental or consequential damages in connection with, or arising out of, the furnishing, performance, or use of these programs.

Registered Trademarks

The IBM Personal Computer is a registered trademark of International Business Machines, Inc.

About the Author



AL STEVENS has been a computer programmer and systems analyst for more than 27 years. His programming career has spanned several generations of computers, from those with vacuum tube logic to the recent personal computer. For the last seven years, Al has been an independent consultant with clients in business, industry, and government. He helps his clients with requirements analyses, proposals, software specifications, and data base design, but he prefers programming. He began using the C programming language in 1982 and now regards it as "the ideal language for the professional programmer."

When he isn't writing software, Al actively pursues interests in music, art, and airplanes. He has been employed at various times as a newspaper cartoonist, a jazz pianist, a dixieland cornet player, and an automobile mechanic.

This is his first book.

Foreword

This book has reusable C language software tools that directly support the development of on-line, interactive software systems. The tools are developed to run on an IBM PC, XT, or AT, but, since the code is written to minimize the few portability problems associated with C, other implementations are possible.

The collection includes file management, indexed data management using B-trees, executive menu management, screen forms management, and the sorting of data, either from file to file or as an in-line function to a program. This book is unique because it provides these capabilities to you, the programmer, in source code form with no royalties or other strings attached. Many examples throughout this book use these tools, and a discussion of the underlying principles applied in their development is included.

These software tools are not mere textbook examples. They are genuine, functioning software modules in use in many real applications around the country. I wrote the programs for use in a software consulting practice. The book came later in reaction to the interest expressed by colleagues and clients in the toolset and its use.

Acknowledgments

Thanks are due to Ron Herold and Pat Thursam who helped me by critiquing early versions of the manuscript. I also wish to express my gratitude to Ted Byrne who made significant contributions to the content and style of the work. A word of thanks goes to Terry Anderson for his guidance and supervision. And special thanks is for Judy Stevens, my wife, friend (and companion). She helped with the editing and logistics, and she was patient.

About the Source Code

The software in this book is used in a wide number of systems in the installations of my clients. Here is a list of some applications where the toolset functions are in use.

1. A data base administrator's package that manages the data element dictionary and data descriptions for a relational data base.
2. A government procurement and contracts financial management system.
3. A multiuser system simulator.
4. The membership and catalog processing for a video tape rental store.
5. A C language cross reference utility package.
6. A government architect and engineering bid specification text management system.
7. A facilities design and construction project management system.

The source code in Appendix A is available to anyone. You can key it in from the listings in Appendix A, or you may prefer to purchase it on an IBM PC-compatible double-sided diskette by mail order. Send $25.00 to:

C Software Toolset
2983 Newfound Harbor Drive
Merritt Island, FL 32952

(Florida residents please add 6% sales tax.)

Preface

We are currently experiencing a phenomenon that has come to be known as the "information explosion," a condition where the availability and demand for information exceeds our ability to process it. This circumstance has been nurtured and fed by the remarkably wide proliferation of small, inexpensive computers; as the potential for information processing has grown, its demand has also grown.

The growth in demand for processed information has spawned a parallel phenomenon called the "software crunch," meaning a shortage of information-processing software. Not enough expensive software is available to feed the inexpensive hardware, and there are not enough programmers to develop it.

This shortage of programs and programmers can be dealt with in two ways: by increasing the number of programmers who are producing code, and by improving the productivity of the ones we already have. The first approach involves the school system. University computer science departments are grinding out graduates as fast as they can, and there is no problem replacing them with new, eager freshmen. The second solution, improving the productivity of the individual programmer, is necessary regardless of how many there are and is the reason the software in this book was developed. Tomorrow's programmer must be more productive than today's.

A lot has been written on the subject of programmer productivity and how to improve it. Most of the research addresses itself to either the problems of managing software projects or the methods and techniques of software design and development. Both views are valid, and both have been seriously examined and expanded upon during the past two decades. Unfortunately, we have yet to correct the dilemma which is characterized by this wry observation found tacked above programmers' desks:

"Good, fast, cheap—select two."

The sad fact is that when we select two, there is no assurance of

a successful outcome; often we cannot get even one out of the three.

Disciplined software development is clearly understood and taught today, but too often large projects ignore the advanced methods developed from years of research and experience. In this book, I will explain those methods that influenced the development of my software tool collection. By example, you will be urged to accept the practices of structured programming, information hiding and, most of all, software tool building. If you do not already know what these things are, then you will learn and not a day too soon.

This book identifies the components of interactive computer applications common among many systems. Applying the disciplines just named I developed those components in a way that will allow them to be reuseable. This constitutes a personal collection of software tools, one that allows a programmer to be more productive. The quantity of code required for a given program is proportionately reduced by the size and scope of the applicable tools from the toolset. The less code you need to write, the more programs you can complete.

This book, then, has four purposes:

1. To promote the use of software tools in the development of interactive systems.
2. To provide a healthy starter set of useful tools for the C programmer.
3. To set an example of orderly and disciplined design and development practices, characterized by reusable tools and the disciplines of structured programming, portability, and information hiding.
4. To provide the reader with motivation and materials to become a more productive programmer.

This is most appropriate on the eve of an era when the attribute of productivity, more than any other, will mark the difference between success and failure in the profession.

Al Stevens

Contents

1 Introduction

This is a book about C language software tools for interactive systems. Interactive systems are systems where a user interacts with the computer to process some information. This book includes C source code for many of the functions common to this environment. It does not try to teach you how to program or how C works; there are other texts for those purposes. It will, however, try to influence your thoughts about programming and style. This book is intended to fill a void that the author has encountered during the development of software systems.

Who is the audience for this book? You are a programmer who will be writing programs for on-line users in an interactive environment. You need to understand the fundamentals of data storage and retrieval and the requirements for data entry and display, and you need a set of software tools that helps you apply that understanding to the creation of interactive software systems.

I begin to fill those needs with this software. At the same time I introduce several algorithms that you might not have seen in source code before, and I set the examples of structured programming and software tool building.

The software tools in this book run on the IBM Personal Computer, but these principles are applicable in other environments. The functions compile and execute by using the Aztec C86 Compiler, the De Smet C Compiler, and the Lattice C Compiler. These compilers are accurate implementations of "standard" C. That is, they implement most of the language as defined by Kernighan and Ritchie in "The C Programming Language." All three compilers come with a nearly complete standard library of

functions. This standard library lets us develop code that has a better chance to be portable—to be moved from system to system.

Some Definitions

In these discussions, we define the person who uses the software you develop as the "user." The "user interface" is the dialogue between the computer and the user. An "application" is the user's use for the system, as in a personnel or payroll application. A software function that calls another function is called the "caller" or the "calling program or function." A "function" is a module of C source code, as in the idiom of the language. A "process," as used here, is a generic use of the computer system within the application, such as the generation of a report.

The Tools

Many software components are common to most on-line, interactive systems. In this book, I specifically address a certain category of application (discussed later). These components are alike enough in their implementations that they can be supported by using a common set of software functions. Let's identify and explain the components. They are:

Menu Management
Data Screen Management
B-Tree Index Management
File Record Management
Record Buffer Management
Sorting

A full set of software functions that adequately supports these components is the beginning of a good software development tool collection. In this book I develop a C programmer's toolset through the use of reusable C language functions.

Figure 1-1 is the system architecture for a software system developed around the C tool functions. The shaded box represents the applications functions; the rest of the boxes are toolset functions. The figure shows the top-down relationships of each of the sets of functions. A function is called by those above it and calls those below it; the cylinders are disk files and the arrows show the direction of data. Suppose that the shaded box is a set of functions to support the interactive processes of membership accounting for an

association or a club. If the system is not very complex, there might be 500 lines of code. The rest of the system is handled by the several thousand lines of code from the C software toolset. The advantages of a tool collection are becoming obvious.

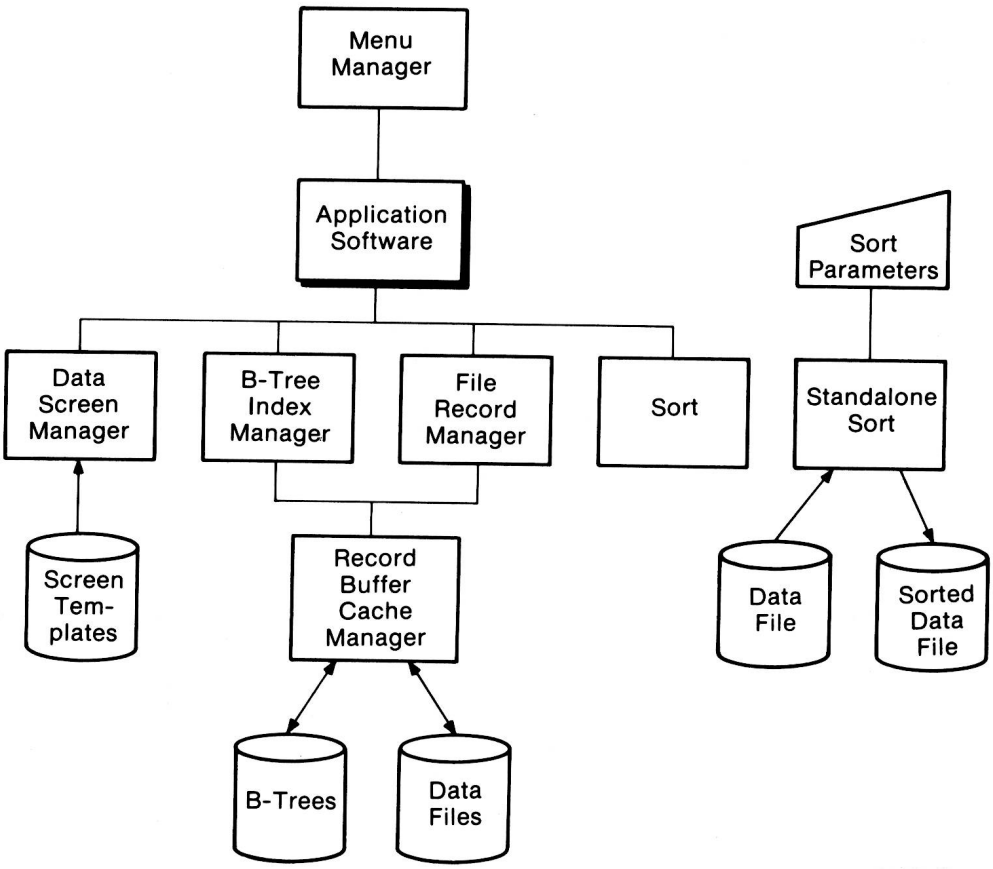


Figure 1-1. C tool functions: system architecture.

Menu Management

Traditional interactive computer systems use menus to allow the user to control the execution of programs. Recently, wonderful machines like the Macintosh have emerged with things called "icons" and hardware "mice" to improve the user-machine inter-

face. These are some great concepts, but the software in this book is not aimed at these techniques. Not yet, anyway. The large majority of today's systems, and, I would guess, the large majority of those to be sold for the next several years are going to use the old-fashioned keyboard and screen. That is the environment supported by the software in this book. That environment uses menus.

Data Screen Management

Most on-line systems communicate with the user by using the video screen, and the user communicates with the system by using the keyboard. The terminal is the medium. In the typical personal computer BASIC program a lot of PRINT and INPUT statements manage this dialogue. These statements represent system prompts and user-entered data items in response to those prompts. It is my goal to replace this BASIC programming technique with a toolset of C functions for controlling the user interface in a consistent manner. The objective is to eliminate a lot of redundant programming and achieve a measure of consistency in the user interface. A library of such functions can be used again and again, thereby reducing the programming effort required for the development of subsequent applications. This is the strength and the common sense of software tool building.

The system developer describes screen formats and provides data collection buffers, and the functions do the rest. Well, almost. Exit points must be provided for the functions to call custom code which does those jobs that are specific to a particular application—special data validation, help to the user, record retrieval, and so on.

B-Tree Index Management

Interactive systems access on-line data from a random access storage device such as a hard disk. To be interactive, this access needs to be fast. It needs to be fast enough that the user doesn't suffer undue delay. Therefore, the computer system must be able to locate a record of data from anywhere in perhaps tens of thousands of records and retrieve it and do something meaningful with it in a very short time. The retrieval is based on some descriptive data entered by the user. For example, a personnel system user may wish to locate the record of a single employee based upon the

employee number. There are a lot of techniques for doing this; different environments can be supported with different techniques. In the interactive environment the records in a file must be located by using various data items at different times. The employee might be located by employee number for one retrieval, by social security number in a different retrieval, by last name in a third, or by department in another. This requires a technique where files can be searched by using the values of more than one data item and where some item values are known to uniquely identify a record (a given employee number is in only one employee file record), while others can be shared by multiple records (the same department number can appear in more than one employee record). The data item used for the search is called a "key." A data item, which is the primary identification for a record (for example, the employee number in the employee file record), is called the "primary" key. A nonprimary key data item that can also be used to search a file (for example, the department number in the employee file record) is called the "secondary" key. Usually, the primary key is unique in a file, and secondary keys can have multiple occurrences of the same value.

A common technique for supporting a data file with multiple keys is the use of inverted indices. An inverted index is another file that is a table of the data item values and is maintained separate from the data file itself. It contains an entry for each value of the data item in the data file and a pointer to the data file record that contains the value. When searched, the index yields the file record address of the relevant data record. The search must be efficient; if there are a lot of entries in the index, its structure must support some technique for rapidly finding a desired entry. The index itself must be maintained. That is, if a data record is added, the index must be updated with an entry to point to the new record. When a record is deleted, the index entry that points to it must also be deleted. When a record is modified, and the change is to an indexed item, the old entry in the index must be deleted, and the new one must be added. To achieve an interactive environment, these index maintenance operations must be fast.

There is a technique for inverted index table maintenance called the B-tree. This is a hierarchical structure of balanced (hence "B-") indices which has the property of providing fast key retrieval and updates. It is maintained external to the data struc-

ture of the file it supports, so it can be eliminated or built at any time without affecting the integrity of the data itself. The tree structure of the index allows a particular entry to be found with a minimal number of key value comparisons. It has the additional capability of providing instant sequential access to the file using the indexed data item. This sequencing can be in ascending or descending order and can start from any record in the file. This often eliminates the need for data file sorting; a given file can be processed as though it were in any of several sequences without the need for an intervening sort pass. This saves a lot of time for on-line retrievals of ordered data.

B-trees are very popular among the developers of data base management system software packages. The B-tree structure has the fascinating ability to appear maintenance-free. Older methods of inverted indexing always seemed to require periodic rebalancing or reorganizing of the indices to maintain efficiency. The B-tree grows and subsides by itself and rarely needs tending.

There must be software functions to build, maintain, and retrieve keys from B-trees. Therefore, this book includes a complete set of B-tree inverted index functions. With these, our toolset expands, as does our understanding of advanced data structures. The B-tree functions represent a significant and complex piece of code. The technique itself is popular; books are available that explain it in-depth, and software packages are available that implement it, but the source code is rarely made available.

File Record Management

When a data record is stored on a disk, it must be written in a location that is known to be available (not already in use) and that can be remembered later when the record is to be read back in. Operating system file managers will allocate physical disk space to a file and provide a directory that allows an application program to find it. Programming languages provide the interface between the application software and the operating system to open and close files and to read and write records. It is the responsibility of the programmer to manage the functional integrity of the data in the records.

The software in this book provides for the description of a file of fixed length records of any size and the random storage and retrieval of a record based upon the record number relative to its