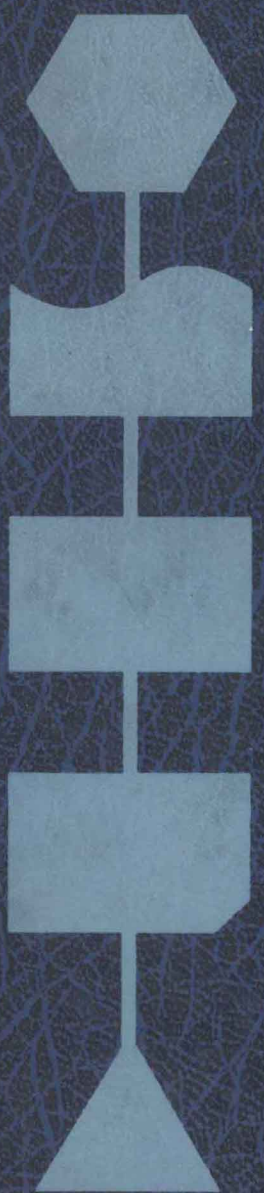


McGraw-Hill PERSONAL COMPUTER PROGRAMMING ENCYCLOPEDIA

LANGUAGES AND OPERATING SYSTEMS

William J. Birnes, Editor

Nancy Hayfield, Production Editor

McGraw-Hill Book Company

New York St. Louis San Francisco

Auckland Bogota Guatemala Hamburg Johannesburg
Lisbon London Madrid Mexico Montreal
New Delhi Panama Paris San Juan Sao Paulo
Singapore Sydney Tokyo Toronto

1 2 3 4 5 6 7 8 9 0 DODO 8 9 2 1 0 9 8 7 6 5

ISBN 0-07-005389-8

Library of Congress Cataloging in Publication Data

McGraw-Hill personal computer programming encyclopedia.

Includes index.
1. Microcomputers—Programming. 2. Programming languages (Electronic computers) 3. Operating systems (Computers) I. Birnes, William J. II. McGraw-Hill Book Company.
QA76.6.M414 1985 001.64'2 85-135
ISBN 0-07-005389-8

McGraw-Hill

PERSONAL COMPUTER PROGRAMMING ENCYCLOPEDIA

LANGUAGES AND OPERATING SYSTEMS

STAFF

Shadow Lawn Press

The Encyclopedia was compiled, typeset, and paged by Shadow Lawn Press, Neshanic Station, New Jersey.

William J. Birnes, President and Project Editor
Nancy Hayfield, Production Editor
Dorothy L. Amsden, Art Director and Illustrator
William Thompkins, Technical Illustrator
Jeanne Neilson, Illustrator
Robin E. Sigmann, Layout
Victoria Boyle, Index

McGraw-Hill

Sybil P. Parker, Editor in Chief
Edward J. Fox, Art and Production Director
Joe Faulk, Editing Manager
Patricia W. Albers, Senior Editing Assistant

CONTRIBUTORS

Jonathan Amsterdam, Artificial Intelligence Laboratory, Massachusetts Institute of Technology.

Henry F. Beechhold, Professor of English and Linguistics, Department of English, Trenton State College.

David Dameo, Director of Computer Operations, RTK Computer Services, Perth Amboy, New Jersey.

Michael Iannone, Chairman, Department of Mathematics and Computer Science, Trenton State College.

CONTRIBUTORS

R. Claude Kagan, Research and Development, Western Electric; Operating Systems Standards Committee, IEEE, Princeton.

Guy Kelley, Chairman, Forth-83 Standards Committee, Forth Interest Group, La Jolla, California.

Yong M. Lee, Professor of Mathematics, Department of Mathematics and Computer Science, Trenton State College.

David Lewis, Department of Mathematics and Computer Science, Trenton State College.

Len Lindsay, President, COMAL Users Group International, Ltd., Madison, Wisconsin.

Kenneth Madell, Software Developer and Programmer, Hamburg, New Jersey.

Lawrence Mahon, Software Consultant and Marketing Representative, Computerland, Inc., Somerville, New Jersey.

Gary Markman, Software and Systems Consultant, Yonkers, New York.

Norman Neff, Professor of Mathematics, Department of Mathematics and Computer Science, Trenton State College.

Ross Overbeek, Research and Development, Argonne National Laboratories, Lisle, Illinois.

Mark J. Robillard, Systems Consultant, Townsend, Massachusetts.

Al Rubottom, President, New Technica, Inc., San Diego, California.

Max Schindler, Editor in Chief, *Electronic Design;* President, Prime Technology, Inc., Boonton, New Jersey.

Stephen E. Seadler, President, Uniconsult, New York City.

Ernest R. Tello, President, Integral Systems, Santa Cruz, California.

Michael Tilson, Director, Human Consulting Resources Corporation, Toronto, Ontario.

Charles R. Walther, President, New Century Educational Corporation, Piscataway, New Jersey.

Linda Weisner, Research Assistant, Department of English, Trenton State College.

CONTRIBUTORS

Michael Whetmore, Professor of Mathematics, Department of Mathematics and Computer Science, Trenton State College.

Robert Wharton, Professor of Mathematics, Department of Mathematics and Computer Science, Trenton State College.

William Woodall, Software Specialist, Somerville, New Jersey.

PREFACE

The concept of personal computing had its beginnings in the late 1970s with the assembly of the first microcomputers based on the technology of the integrated circuit. The initial commercial success of the microprocessor-driven units converged with the ongoing development of high-level computer programming languages. The appearance of these languages in the 1950s took programming, once a field open only to the engineers and designers who had originally developed computers, out of the laboratory and into the business marketplace. By the late 1960s there were a number of high-level languages with applications in the sciences, engineering, business, and even elementary and secondary education. When the first BASIC interpreters were bundled with the new "personal computer" packages in the 1970s, the two development streams were officially joined, and the personal computing revolution began.

Over the next eight years, each success in personal computing technology prompted a new surge of development. The machines themselves grew in power and capability from 8-bit 16K PET computers with membrane-type keyboards and onboard cassette recorders to the IBM PC-AT with its 20 megabytes of hard-disk storage and true 16-bit speed. By the end of 1984, the LISA technology developed by Apple for its 32-bit office machines was repackaged to appeal to a home market and quickly caught on as the Macintosh. In a lockstep with the development of new hardware systems was the invention of new software systems and applications programs. The BASICs of the 1960s and 1970s gave birth to more powerful versions that had built-in graphics and music commands and system utilities that allowed even novice programmers a sophistication and efficiency of code that previously could only have been found in assembly language routines. However, beyond BASIC, versions of Pascal, C, and Forth were developed which put enormous programming power into the hands of personal computer users and allowed them to emulate the processing capabilities of large mainframes at their desktop terminals. And this is only the beginning. Languages such as Ada, the Department of Defense's new projected standard information-processing language, and Prolog, which is at the center of artificial intelligence research and development, have recently been implemented on personal computers and will become more popular as succeeding generations of more powerful computers find their way into the home and business markets.

This proliferation of computing language implementations has created a serious need for a single reference volume which not only introduces the various languages and indexes all of their command words and statements, but provides for a cross-referencing of applications and a comparison of the languages' capabilities. This is the purpose of the *Personal Computer Programming Encyclopedia*. The Encyclopedia illustrates the capabilities of each language with overviews of the language's design and architecture, and by comparing the operational differences of each language through sample programs, the Encyclopedia demonstrates the different ways applications can be addressed by programmers working within the different language environments.

The Encyclopedia is divided into two types of sections: the double-column sections which cover the high-level languges, operating system commands, and assembly language commands, and the single-column sections which contain background and introductory material. In the single-column sections, readers will find articles which explain the architecture and design of computer programs, examine the user-oriented issues of software development in business

PREFACE

and education, and explore the newest areas of development in graphics, robotics, and artificial intelligence. In choosing these different types of entries for the Encyclopedia, recognition has been given to the major areas of personal computing that affect users: (1) the need to understand the logic of program design; (2) the current trends and issues in the most important areas of software development; (3) the diversity of languages that are available to personal computer users and the primary applications of these languages; and (4) the relationship of hardware systems to software systems. The result is a volume that addresses the needs of the entire personal computing community from the business user and consumer of professional software products to the home user learning about a type of information technology that promises to transform the ways people organize their lives.

The Encyclopedia provides background histories of the high-level programming languages, operating systems, and applications software cited. It relates the development of these various software tools to the current computing environment as well as to the historical period during which the software was originated and marketed. The result of this approach is a social history of personal computing in which programmers, personal computer users, and general readers will discover the underlying reasons for much of the product development in the marketplace. The Encyclopedia explains the different trends in languages, operating systems, and applications software over the past five to ten years, and the effects of these products on users in different professional fields.

The Encyclopedia contains an index to all of the keywords and statements in the high-level languages that are cited. This index is an important reference tool for programmers and serious users because it provides an immediate cross-reference between the different languages. Programmers seeking to translate source code from one compiler to another or from one dialect of BASIC to compiled or structured BASIC will find the cross-index a handy tool. General readers interested in the history and intellectual backgrounds of programming languages will find in the cross-index of high-level language keywords a generic approach to the types of commands that are used in programming. Teachers will also find this system a valuable reference tool for use in comparative programming.

The Encyclopedia also examines the different corporate cultures from which the most popular types of personal computers have evolved and evaluates the dynamic relationships between manufacturer and the manufacturer's product history, the hardware system and supporting software, and the user market the computer was targeted to reach. Readers will find an interesting perspective on the current trends of technological development in the areas of hardware, software, and operating environments. There is a capsule summary of the history of personal computers from the first attempts to market basic user-assembled kits to the 32-bit supermicros that will be making their appearance within the next several years. Their history, brief as it is, will provide a needed background to the dynamic microcomputer marketplace and the different products that are announced in the computer magazines and newspapers.

PREFACE

The articles in the *Personal Computer Programming Encyclopedia* are written by individuals from a variety of backgrounds. This diversity of opinion is reflected in the different levels of emphasis within the entries and the broad perspective of the volume in general. In short, the Encyclopedia embraces the types of related informational materials that spread across the traditional boundaries often found in a reference book on science and technology. This is what makes the volume an innovative reference tool.

While a number of computer dictionaries and comparative reference books on programming languages have appeared recently, the *McGraw-Hill Personal Computer Programming Encyclopedia* is the only single-volume reference to provide a comprehensive introduction to the entire personal computing environment both as a science and as a commercial industry. Thus, it will become a valuable reference both for the computer professional and for the novice. Business users, students, teachers, hobbyists, and home users will find this *Encyclopedia* a most useful desktop computer reference.

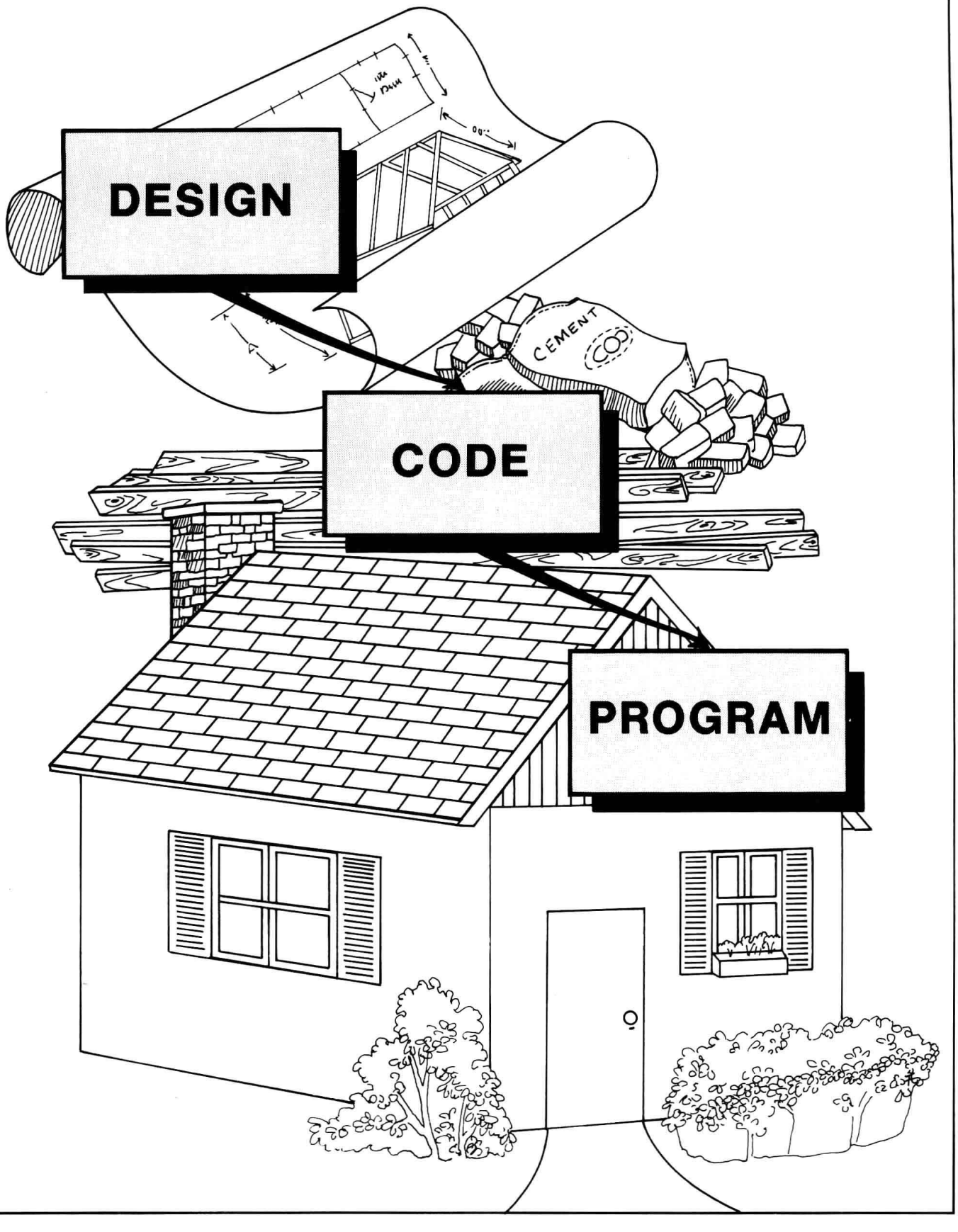
William J. Birnes
Editor

McGraw-Hill

PERSONAL COMPUTER PROGRAMMING ENCYCLOPEDIA

LANGUAGES AND OPERATING SYSTEMS

How to Begin to Program



Dorothy L. Amsden

CONTENTS

CONTENTS

CONTENTS

8

1

PROGRAM DESIGN AND ARCHITECTURE

As in most creative work, the fundamental aspect of writing good applications and systems programs lies in the preliminary design and architecture. It is on this level that some of the most important thinking takes place. Goals are defined, pathways are mapped out, logical relationships between data types are developed, and the rules that will govern the decisions the machine will make are stipulated. It is here, at the very heart of a well-designed program, that a definition of truth is implemented, and the execution of this program, the processing of line after line of code, is a test for that truth. And as a statement of truth and a test for that truth's existence in the data that flows through it, the computer program takes its place right alongside literature and art as a form of creative expression. There is a practical creativity, to be sure, but a computer program is no less creative and structured in its disciplined expression of truth than a line of poetry by Keats, a portrait by Albrecht Dürer, or a Bach concerto.

What takes place in the design and architecture of a computer program? On the most obvious and visible level, it is a patient sequence of logic that expresses the complexity of human thought in terms of an organized pattern of connected decisions to be implemented ultimately as a series of electronic pulses. On an even more fundamental level, it is nothing less than the definition of the reality that the computer will understand as its truth. The design of a computer program, therefore, is a microcosm of the physical universe, and as an amalgamation of artistic creativity and technical precision, it is the marriage of C. P. Snow's "Two Cultures."

The choice of the actual programming language and the subsequent coding of the program, while important, take place after the logic of the program has been designed. Programming languages by themselves are only forms of machine code generators. By definition, they are a source of code that is either compiled or interpreted by the machine and translated into the sets of instructions that the machine can process. As a source of code, they help the programmer implement a coherent and executable logical design and serve as a matrix for the actual commands. In addition, high-level computing languages provide for the definition of the types of data and, in some cases, particularly COBOL, the specific machine environments. But, while an indicator of program efficiency, the programming language is not, and should not be considered, the ultimate indicator of quality. The best programs are good, not because they are written in C rather than BASIC, or in LISP rather than Pascal, but because they are

designed from concept through code to be disciplined and creative tests for validity and truth.

As an implementation of a logical structure, the original programming environments in the early days of digital computing were required to be designed efficiently and completely because they were written on a machine level which was unforgiving of mistakes. High-level languages, which were developed later, were oriented more toward the natural language of speakers than they were toward the opening and closing of electronic circuits. High-level languages addressed compilers and interpreters which in turn generated the machine code. As a result, high-level languages often have built-in mechanisms which, though quick to trap errors in the usage and syntax of source code, can sometimes be quite forgiving of fundamental mistakes in design and logic. And it is these hidden structural mistakes which ultimately surface in the program's execution to make debugging the program a seemingly impossible task. Therefore, all professional computing training curricula, whether on a secondary school, college, or vocational level, usually begin with a unit on program design and architecture.

But program design and architecture begin with an understanding of commonsense sequential logic. In other words, to design a program, an individual need not have the actual high-level language code at his or her fingertips; rather the person must understand fundamentally how the computer of choice will operate and how to define logically the complete task of the program from beginning to end. It is this task definition and the realistic design of an operation from beginning to end that is necessary in order to write a good program. Whether the task is designing an operating system or writing a program to calculate the principal and interest payments on a loan, the programmer begins with a list of items the program must accomplish and ends with a chart showing the order in which the program will do just that. This section will introduce you to the elements of programming design and architecture from a programmer's point of view and will look at the elements of goal-setting, evaluating the programming tools to be used, and the construction of a programming structure.

Principles of Program Design

In principle, designing a program is not different from approaching any other task. We require a goal that is clearly defined because we need to know what we want to accomplish, we require an understanding of our beginning resources, and we must have a thorough understanding of the means we will use to reach the goal. As an example, consider the act of going to work in the morning. The goal is to get to work. The beginning resource is your home. The means to reach the goal is some form of transportation. The only difference between this process and the process of writing a computer program is that there's no need to make any part of the task of getting to work in the morning a conscious, consistent, and repetitive test for truth every day. However, how would these actions appear without a goal? Your normal activities such as waking up at 6:30, getting dressed in work attire, going to the train station, and so on, would seem silly indeed if you performed them on a Sunday morning or a holiday. The point is that most of the things we do need a goal in order for the actions to have any meaning. Designing a program is no exception.

Probably the least practical way to begin writing a computer program is to sit down at your computer and start writing code. Many beginning programmers, eager to see results, will do just that. The result is usually very awkward and inefficient code, poor documentation, and an absolute nightmare when debugging time comes (and it always does).

The three steps named above—goal, beginning resources, and means—are essential in the design of a good program. And the first language any beginning programmer should consider using is not BASIC, Fortran, Pascal, or COBOL, but rather English. Specifically, the way to begin writing a program is to identify