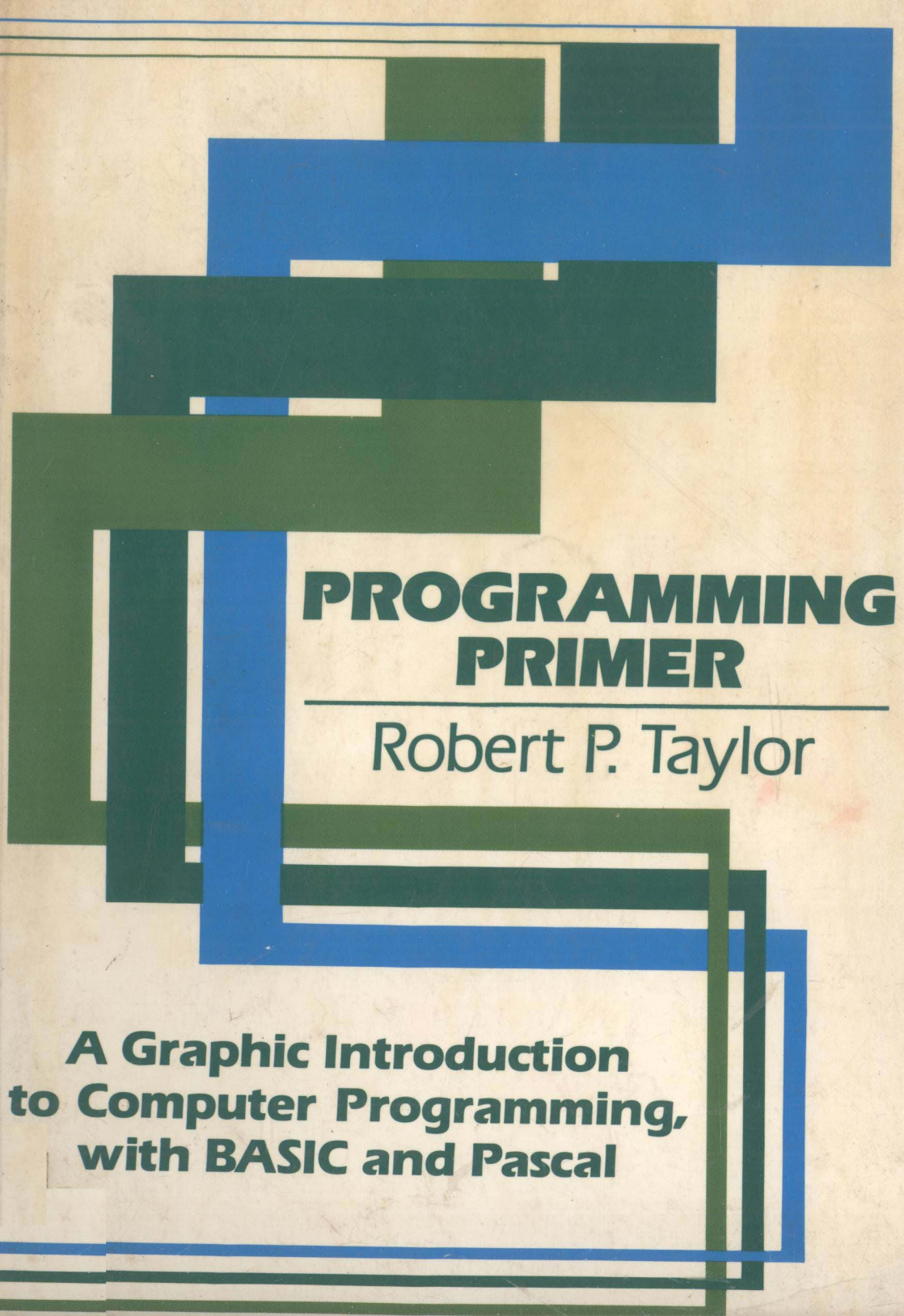

PROGRAMMING PRIMER

Robert P. Taylor

A Graphic Introduction to Computer Programming, with BASIC and Pascal

PROGRAMMING PRIMER

A Graphic Introduction to Computer Programming, with BASIC and Pascal

Robert P. Taylor
Teachers College, Columbia University

ADDISON-WESLEY PUBLISHING COMPANY
Reading, Massachusetts
Menlo Park, California
London • Amsterdam
Don Mills, Ontario • Sydney

Library of Congress Cataloging in Publication Data
Taylor, Robert P.
Programming primer.
Includes index.
1. Electronic digital computers—Programming.
I. Title.
QA76.6.T393 001.64'2 81-2209
ISBN 0-201-07400-1 AACR2

ISBN 0-201-07400-1
ABCDEFGHIJ-AL-898765432

PROGRAMMING PRIMER

preface

GENERAL COMMENTS

This book is designed to introduce the process of computer programming to those with no previous knowledge of the subject. The text assumes no math background beyond the introductory secondary school level. Solved examples introduce the major facets of programming and focus on how to develop a problem solution suitable for computerization. A major area of concentration is the student's development of an appropriate approach to formulating problem solutions by means of FPL (First Programming Language), a well-structured programming language designed specifically for this purpose at Teachers College, Columbia University. Complete mastery of FPL requires no access to computers of any sort. Yet because FPL was designed to embody the best of current thinking on programming by computer professionals, mastery of FPL leads naturally to the successful use of many well-known computer languages. The book starts the student on the path to learning two such languages: the text demonstrates how solutions formulated in FPL may be translated directly and mechanically into well-structured programs in BASIC and Pascal.

WHO CAN USE THIS BOOK?

This book can be used successfully as a first text in programming at any of a number of levels. Though originally developed for educators, it is equally suited for others as well. Those who find the more mathematical approach adopted by

most other programming texts to be a formidable barrier to getting started in programming may find this book a welcome alternative. At the same time, though it does not presuppose a mathematics background nor use much mathematics, it is a thorough introduction to the basic logic structures involved in all programming. It therefore can also serve admirably as an introduction to structured programming for students in engineering, the hard sciences, and computing. Because programming is primarily well-organized thought translated into instructions for a computer, all computing requires the same first achievement of the novice—learning to conceptualize a problem in a form suitable for computer solutions.

All kinds of students have used this material successfully, both as a class text and as an independent-study guide. Students using it have ranged in age from 12 to 60, in experience from none to extensive, and in occupation from housewife to computing professionals. They have come from developing as well as developed countries around the world.

The material has been used both as a review text and as a "first look" text. Whenever students are being introduced to computing with minimal help from instructional staff, this is a reasonable text to start with—whether the students are working completely on their own or loosely attached to a class. The text is designed to forestall their learning a lot of bad habits of structuring when they pursue programming on their own. Attention focuses on the organization of programming rather than on myriad details of hardware, languages, monitors, or operating systems. Study of this book profited a number of students who already knew considerable programming but who had learned eclectically and lacked a sense of good programming structure.

HOW LONG SHOULD IT TAKE TO COVER THE MATERIAL IN THIS BOOK?

Though this text should prove equally suitable as an introduction to programming for all beginners regardless of their backgrounds or eventual objectives, though the principles presented must be mastered by anyone who would learn to program adequately, the speed with which the beginner can progress will depend on the particular background and objectives of the learner. Thus the text may be covered in the initial few weeks of courses designed for engineering and science students or in a few weeks of independent study by learners with considerable experience in logic and in organized thinking. Or it may be the basis for a full term or even a full year's work for those without much background in formal problem solving. Testing the material of the book with a wide range of learners has indicated that the type of previous educational experience, not educational level, is the best predictor of how fast the material will be mastered.

A NOVEL PRIMER

This text is novel in four respects. First, it relies heavily upon graphics to present the fundamental ideas of programming. Second, it goes into considerable detail and is therefore quite long for an introduction or "primer." Third, it requires no access to hardware to study and master most of the principal ideas. Fourth, it embodies a multilingual approach.

The Graphic Presentation of Programming

The reason for using a graphic presentation in FPL is pedagogical. Human beings use graphic representation constantly, are familiar with symbols of all kinds, and use them constantly to organize ideas of every sort. Till now, with the possible exception of von Neumann flowcharting, little use has been made of graphical approaches to teaching programming. Instead, texts and manuals have emphasized the use of written character languages exclusively and have emphasized exactly how these written languages look, in the form *computers* find it convenient to display. Everything is thus reduced to sequential listings of "statements" from the particular language. However, human thinking is seldom actually purely sequential. Human beings often think in terms of hierarchies and parallel structures. Therefore the natural organization of thinking is *not* solely in the form of a sequential listing of ideas, the form a computer finds most convenient. FPL as used in this text departs from this traditional approach to teaching programming and deliberately takes advantage of not being bound to sequential display. Parallel thoughts are represented side by side. Alternatives are shown in appostion. Hierarchical thinking is encouraged immediately.

The rough thinking essential to solving any but the simplest problem is encouraged by including a formal representation of rough ideas, the EPISODE, as one of the primary FPL constructs. The graphic approach makes this inclusion simple and natural, and students are consistently encouraged to represent rough ideas in successive drafts of each program. Through both the model program solutions and the exercises, students are thus encouraged to see problem solving as the processs it is, rather than to think mistakenly that it is an inspired, instantaneously realized event.

The Length of Detail of This Primer

This primer is long because it assumes that few things are self-evident to the novice, that the "QED" or "clearly, it follows from here" approach that produces short texts and that leaves to the imagination much of the detailed logic

of a program or a problem solution is not a fruitful approach for teaching new material to the uninitiated. Many explanations are therefore presented in great detail, with illustrations of intermediate stages. The approach assumes it's better to present explanations long enough to help those who need them than to omit such detail. If present, the detail can always be skipped over by those who don't need it; if omitted, it is not available to those who do need it. The extensive presentation of examples is also based on the idea that learning occurs by modeling; thus every chapter concludes with a complete program embodying the major idea of the chapter.

The Hardwareless Aspect of This Approach

The material in this book depends on access to hardware only for the BASIC and Pascal interpretations—the primary presentation in FPL requires no computer. This approach has been taken for several reasons. Because there is considerable general educational benefit involved in learning how to program and because computing hardware is not always directly available to many who might profit from this learning experience, one major motivation for developing FPL was to develop a sound way to teach programming even in the absence of access to hardware. At the same time, even where hardware is readily accessible, there are strong advantages to mastering basic structural concepts before using that hardware extensively. The overall and most important thing to learn is *not* a collection of eccentric details about a given language or piece of computer hardware but how to move from a problem to a well-organized solution for that problem. Trying to learn how to solve problems by organizing things while simultaneously trying to learn all the transitory eccentricities of a language designed to "fit" a particular piece of hardware is at best pedagogically questionable. Such simultaneous approaches have often contributed directly to the unfortunate situation now prevalent in computing generally: great quantities of programmer time are spent trying to fix up and change programs that were poorly conceived and badly written in the first place.

The Multilanguage Approach

While some courses using this material have taught only the FPL and ignored the BASIC and Pascal interpretations altogether, it is more usual to have the students learn one or both of the other languages along with FPL. Whether the material on BASIC or Pascal should be introduced gradually, in parallel with the FPL (as the text presents it), or presented in some more concentrated fashion (e.g., after all FPL has been learned) is a matter of choice for the learner and the teacher.

The two sets of interpretations are designed to provide a simple, mechanical technique for translating FPL solutions. Neither set is intended to be a complete text suitable for fully mastering that language; each set is only a carefully designed springboard into its appropriate use. Teachers can be of immense help in guiding the student into these languages, but the student can also make the transition alone, provided appropriate reference manuals and documentation on how to use each particular language in the local environment are available.

SUMMARY

The major objectives of this book are to teach students how to generate problem solutions suitable for computerizations, to give them a strong sense that programs are written to be intelligible to other people, and to convince them that good program structure is an essential characteristic of any intelligible program. To achieve these objectives, the programming *process* is presented in considerable detail, using FPL, a language specially designed for just this purpose. That language distills the essence of programming into a minimum of graphically represented components the student can easily remember and mechanically translate into other computer languages. The translation languages included are BASIC and Pascal.

January 1982 R.P.T.
New York City

contents

Chapter 5
A Row of Similar Data Items—
The One-dimensional Array

Chapter 6
Rows of Similar Data Items—
The Multidimensional Array

Chapter 7
Synthesizing—
Creating a Survey Analysis Program

Chapter 8
Forcing the Computer to Choose —The EITHER

Chapter 9
Two Embellishments— Compound Assertions and ENDFILE

Chapter 10
Keeping Programs Easy to Read, Repair, and Modify— The Program Block

chapter

1 2 3

4 5 6

7 8 9

10 11 12

Fundamentals

INTRODUCTION

Computer programming is the process of instructing computers to carry out specific tasks. The instructions are called *programs* and the "language" in which the set of instructions is cast is called a *computer language.*

Most computers perform only a limited number of specific electrical actions. A computer language puts the computer's electrical activity at the command of the programmer. The programmer instructs the machine by linking a set of instructions together so that the computer produces the actions desired by the programmer when it reads the instructions. This means, of course, that the computer program must be put into a machine-readable form so that the computer, via one of the external devices attached to it, can read the program. By feeling or seeing the patterns of holes punched into a series of computer cards or by sensing the electrical signals generated by keys on a typewriter-like terminal, the computer can translate a program into electrical actions. The actual transformation of the instructions into electrical actions is called running or executing the program.

There are many different languages and many different computers. Some languages run on many different sizes, makes, and types of computers; and some run on only one. Some languages are relatively old and include "fossil" components representing computer hardware requirements that no longer exist; some languages are so new that only a few programmers have yet learned them. *All* languages have their eccentricities and *none* is implementable on all computers or known to all programmers.

This book is based on FPL or *First Programming Language.* It is given this title because it is designed to be used as the first programming language taught to a beginner. Although FPL incorporates features common to various languages, it does not actually run on any computer.

The design of FPL is based on the assumption that a general approach to programming should be independent of the details of particular languages. It is better to learn programming without worrying about the inevitable eccentricities and exceptions of a particular machine and a particular language. Once this approach has been mastered, the student should have no trouble moving from FPL to other languages.

The components of FPL will be introduced gradually. In this chapter we will introduce several, beginning with one component crucial to all program development. It exemplifies the idea that the creation of any computer program is an extended process.