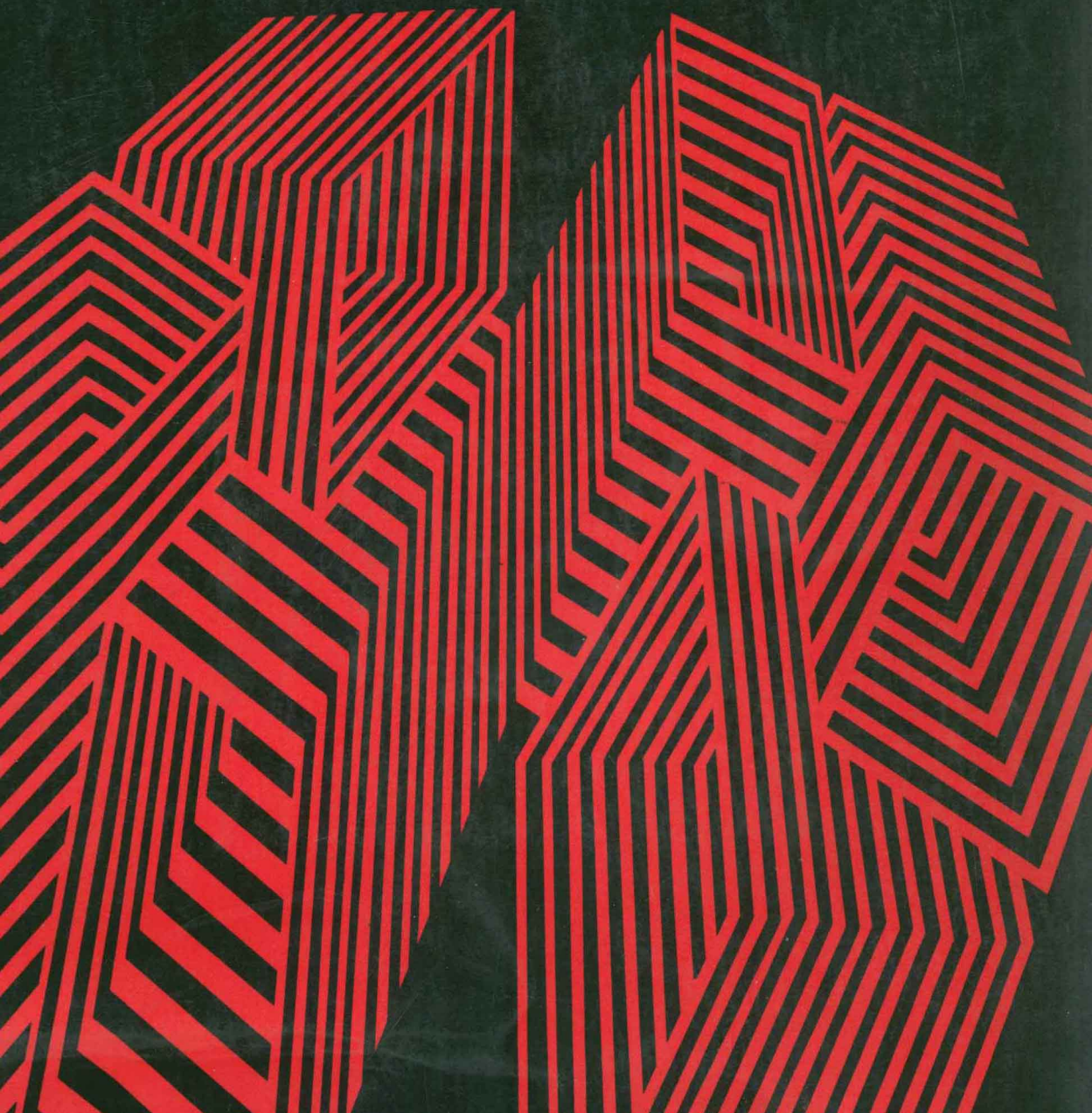
John B. Moore

PASCAL

Text and Reference

Second Edition



Pascal™: Text and Reference

second edition

John B. Moore

Reston Publishing Company, Inc.
A Prentice-Hall Company
Reston, Virginia

To Barb

- a very special person

ISBN 0-8359-5440-4

10 9 8 7 6 5 4 3 2 1

Printed in the United States of America

Preface to the Revised Edition

This revised edition contains a number of enhancements and updates which reflect the evolutionary nature of modern computing.

One important change is the use of the keyboard as the default source of input data in the example programs. This change is a reaction to the growing number of personal computer users and the proliferation of interactive applications.

Appendix C has been rewritten to provide a complete description of the Waterloo Pascal implementation of the language. It reflects the latest specification of the compiler. Waterloo Pascal is being used by an ever-increasing number of educational institutions and individuals because of its comprehensive facilities, superb diagnostics, and excellent reliability.

Other improvements include: the addition of many new programming problems; over one-hundred new index entries; a consistent approach in the algorithms used in Chapter 14; and more comprehensive explanations of input-output operations in Chapters 2, 5 and 12.

I would like to acknowledge the helpful suggestions of many who have used the first edition of this book. Valuable advice concerning the typesetting of the material was received from Bruce Uttley. The material in Appendix C is taken in large part from section G of *Pascal Reference Manual and Waterloo Pascal User's Guide* by Boswell, Welch and Grove published by WATFAC Publications Ltd., Waterloo Ontario. The permission of the authors and the publisher to use this material is acknowledged and appreciated.

Waterloo Ontario Canada — John B. Moore

Preface

Pascal is rapidly becoming one of the most widely used programming languages in the world. This growth is due to a number of factors most important of which is the discipline it imposes upon the programmer. The rules of the Pascal language, coupled with good programming style, result in good programs.

Goals. The purpose of this book is to describe how to use the Pascal language correctly and effectively. It is a comprehensive description of standard Pascal although unique features of three particular implementations of the languages are summarized in the appendices. It is intended for the person who wants to make a systematic study of the language and who, when finished, requires a solid reference.

Assumptions. It assumes the reader has no previous programming experience. Readers already familiar with programming concepts and/or another programming language can quickly read those paragraphs and explanations related to algorithm development and focus on the attributes of Pascal. Problems and exercises are taken from a wide variety of subject areas.

Pedagogy. The examples chosen to illustrate each component of the language are as simple as possible. In this way the reader can focus on the programming concept without becoming enmeshed in the in logic needed to solve the illustrative problem. The examples are complete programs. All have been run and listed using the Waterloo Pascal processor which accepts standard Pascal.

The material is organized so that the reader may proceed sequentially through the entire text. Each topic is presented using a four-step sequence.(1) Here's what we are trying to do; (2) Here's how we can do it with our existing knowledge; (3) Here's a better way to do it; (4) Here's what we did. Each chapter begins with questions which the chapter answers and concludes with a summary of the key concepts and important programming skills presented.

Exercise questions are found throughout. These test the reader's understanding immediately following the presentation of new material. In the author's view, these questions are one of the most efficient ways to imbed new knowledge. A variety of programming problems is found at the end of each chapter. They range in difficulty from very simple to complex. Some involve simple mathematical peculiarities which stimulate the reader's curiosity. Others are designed to massage the programming knowledge developed in the chapter.

Throughout the book, emphasis is placed on good programming style. Indentation rules, naming conventions and commenting guidelines are explicitly stated and consistently followed.

Organization. The book is divided into three parts: Statements and Values, Data Structures and Dynamic Variables, and Appendices.

Part I describes the statements which are used to process the three basic kinds of values - numbers, characters and Boolean (true-false) values.

Chapter 1 describes the five-step sequence which is followed when a computer is used to assist the problem-solving process. The first example results in a complete program which the student may enter and run. Sufficient programming detail is presented in the three examples to allow a reader to solve a wide variety of simple problems using the examples as prototypes. Attention is devoted to the acceptance of errors and the understanding of diagnostic messages.

Chapter 2 begins the rigorous study of numeric processing. Constants, variables, value assignments and simple input-output are covered in detail.

Chapter 3 describes Boolean values and operations and then shows how they are used in the decision-making statements of the Pascal language.

Chapter 4 illustrates, compares and contrasts the three statements used to control loops.

Chapter 5 explains character and text processing. It also summarizes input-output using READ and WRITE with the standard files INPUT and OUTPUT. The difference between numeric and non-numeric input-output is one of the most difficult parts of the language to learn. Consequently, detailed examples and explanations are given.

Chapter 6, extends the readers knowledge of types to include enumerated and subrange types. A summary of the processing rules for all scalar types is provided.

Chapters 7 and 8 describe functions and procedures respectively. Functions, being simpler, are explained first. Particular attention is paid to rules of scope and the meaning of the terms global and local identifiers. Chapter 8 shows how procedures differ from functions both in form and purpose. The last section of the chapter gives guidelines for partitioning algorithms into procedures. It also suggests criteria for choosing the nesting structure of blocks.

Part II of the book begins with an overview of data structures in general and Pascal data structures in particular. Differences among arrays, records, sets and files are emphasized.

Chapters 10 through 13 describe the four predefined structures available in the language. Arrays, having the most utility, are described first. The material describing records and files builds on the knowledge already gained. Sets are described in Chapter 13. An understanding of other data structures is not necessary to use sets and this material can be studied independently of arrays, records and files.

The final chapter describes the use of pointer variables. Individuals not familiar with memory concepts and indirect addressing often find the use of pointers one of the more difficult programming ideas to grasp. Consequently, a very simple example is presented early in the chapter to show the mechanics of their use. This leads to a discussion of the two most common kinds of data structures requiring pointers, namely linked lists (of which stacks and queues are specify instances) and trees. Examples are provided which demonstrate the use of binary trees in information retrieval applications.

Part III of the book contains six appendices. The first two provide a reference for character sets, standard identifiers, operators and the syntax of the language. Appendix C describes the implementation dependencies of the Waterloo Pascal compiler. Appendices D and E summarize those features of Pascal/VS and IBM Pascal for the IBM Personal Computer which are not part of standard Pascal. Appendix F gives a number of suggestions for reducing debugging time and for improving memory and execution-time efficiency.

No one writes a book without a lot of help. I would like to thank Kay Harrison for her excellent work in entering the original, readable(?) draft of the manuscript and for making the innumerable changes in subsequent versions. Mike Ruwald designed the Script macros used to format the text. Jim Dodd found many typos and made excellent editorial suggestions. For errors that may yet be present, I take full responsibility.

To you the reader, I hope you find this a useful and enjoyable book. I have tried to keep your needs in mind from start to finish.

Waterloo, Ontario Canada

John B. Moore

Preface to the Student

This is a book that teaches you how to speak a language – a language which permits you to use a computer to help solve problems. The language is called Pascal. It was originally developed by a professor to help students learn the principles of computer programming. Because he designed the language so well, it has rapidly become one of the most widely-used languages in the world.

Reading a book about computer programming is like reading a book about playing the piano – there is only so much you can learn without intense practice. To become fluent in the use of the Pascal language, you must practise, experiment and test your knowledge continually. Good luck and good programming.

Part I:
Statements and Values

TABLE OF CONTENTS

CHAPTER 1: GETTING STARTED

Questions Answered in this Chapter:

1. What is a computer?
2. Why do we use them?
3. How do we use them?

1.1 What Can a Computer Do?

Computers are attributed with many remarkable powers. However, all computers, from the large ones used to control the space flights to the micro-sized ones you can hold in your hand, are simply collections of electronic components which have three basic capabilities.

First, they have circuits which perform the four basic arithmetic operations denoted by the symbols + - * /. An asterisk or star indicates multiplication; two times three for example is written as 2*3. The slash symbol represents division. For example, the value of 7/5 is 1.4. When two integers are being divided, you can request that "integer division" be performed meaning that the remainder is ignored. Integer division is denoted by "DIV". Therefore the expression "15 DIV 4" has a value of 3, not 3.75. Chapter 2 contains a detailed description of arithmetic operations.

Second, computers have circuits which make decisions. Their decision making capabilities are not such that they can answer a question such as "Will it rain next Tuesday?" or "Who would win a war between Russia and China?" The decision making capabilities of a computer are limited to deciding

1. if one number is less than, equal to, or greater than another number
2. if one character (letter, digit or symbol) comes before, is the same as, or comes after another character in dictionary order

Third, computers have circuits for performing input and output operations. That is, they are able to accept input in the form of instructions and data from human beings. (By data we mean the numbers and characters manipulated by the instructions.) And computers would still be useless machines if we were unable to get the results out of the computer in a form that humans can understand. Thus computers have circuits for sending signals to printers and display screens.

Since all computers have only these three limited capabilities -- arithmetic, decision making and input-output, it is only natural to ask the following question.

1.2 Why Do We Use Them?

There are three reasons why computers are so widely used.

Speed. First, computers are fast. How fast is fast? A powerful computer can perform several million instructions per second. If you or I were to do five million additions of ten digit numbers it would take us -- working a forty hour week -- about four years. Speeds for decision making are also measured in MIPS (millions of instructions per second) but speeds for many input and output operations are thousands of times slower because mechanical motion of cards and paper is often involved. A typical high speed printer for example prints 1000 lines of output per minute. Information can be sent over a standard telephone line at about 10 words per second -- about 1/25 of the rate of a high speed printer.

Accuracy and Reliability. In spite of newspaper headlines such as "Computer Fails Student", incorrect statements from credit card companies and other horror stories of computer foul-ups, you and I realize that it is seldom the machines that make mistakes -- it is the people who make the errors. Computers are remarkably accurate and operate for months, performing billions of operations without an error! Because they are man-made, they occasionally break down and have to be repaired.

A Big Problem = A Set of Little Problems. The most important reason computers are so widely used is that almost all big problems can be solved by solving a set of little problems -- one after the other. If each of these little problems is so simple that it can be solved using the limited capabilities of the computer, then we end up solving the big problem. For example, doing the payroll for a large corporation is indeed a big problem. But in order to solve this problem we need only do the following kinds of things for each person on the payroll: First, input information about the employee such as hours worked, rate of pay, etc. Second, do some simple arithmetic and decision making; Third, output a few lines on a check. By repeating this process over and over again, the payroll will be finished. Since computers can do all of these operations quickly and accurately, the reason for using them is obvious.

Having seen what these machines can do and why we use them, let's find out the steps that are necessary to have them assist us in solving a problem.

1.3 The Five Steps in Problem Solving

There are five steps which must be followed when a computer is used to help solve a problem. They are:

1. Define the problem

2. Develop a procedure for solving the problem.

3. Translate this procedure into a language the computer understands.

4. Enter the instructions and data into the computer.

5. Tell the computer to execute the instructions.

Note that the computer isn't even involved until step 4! In fact, only in step 5 does it operate without human intervention. We shall look briefly at what is in involved in each of these steps and then study an example.

Step 1: Define the Problem. Unless the problem is well-defined, there is no sense even thinking about using a computer to help solve it. The people who get paid the highest salaries in the computing business are those who are trying to answer the question "What, precisely, is it that we want to use the computer to do?" Glib answers such as "do inventory control" or "type form letters" are no good. The problem must be well-defined. Before you write any program, ask yourself "Do I know exactly what the problem is?"

Step 2: Develop a procedure for solving the problem. A word which means "a procedure for solving the problem" is *algorithm*. An algorithm takes the form of a sequence of instructions which, if followed by a moron, will solve the problem. Most algorithms in this book are relatively simple.

Because an algorithm processes numbers and characters it is also necessary to describe the *data* (the objects manipulated by the instructions).

Step 3: Translate the procedure and data descriptions into a language the computer understands. The result of this process is called a computer program. There are literally hundreds of languages which computers can "understand". The programming language described in this book is called Pascal in honor of the sixteenth century Swiss mathematician by that name. It was originally invented to teach the principles of computer programming. Because it is so simple and so powerful, it has rapidly become used for all kinds of programming problems. Like natural languages, programming languages tend to grow (verbs, nouns and sentence structures are added to the language). Pascal is no exception.

Nonetheless, computers can really only understand one language. It consists of long strings of ones and zeros called bits. A *bit* is a *B*inary dig*IT*, namely a zero or a one. All programming languages are translated into strings of bits by a special program called a *compiler*. More will be said about the compiler in Step 5.

Step 4: Enter the program into the computer. Once the algorithm and data descriptions have been written in a programming language, you must enter the computer program and data into the computer. This is done in one of two ways. The lines in the program can be punched into cards, one card per line.