PROGRAMMING

STANDARD PASCAL

R.C. Holt
J.N.P. Hume

PROGRAMMING STANDARD PASCAL

R. C. Holt
J. N. P. Hume

Department of Computer Science
University of Toronto

RESTON PUBLISHING COMPANY, INC., Reston, Virginia
A Prentice-Hall Company

Library of Congress Cataloging in Publication Data

Holt, Richard C
Programming standard Pascal.

Includes index.
1. PASCAL (Computer program language) I. Hume, J. N. P., joint author. II. Title.
QA76.73.P2H64 001.64'24 80-456
ISBN 0-8359-5691-1
ISBN 0-8359-5690-3 pbk.

10 9 8 7 6 5 4 3

Printed in the United States of America.

PREFACE

This book is intended to form the basis of an introductory course in computing. No particular mathematical background beyond basic arithmetic is assumed; examples are taken largely from everyday life. In this way, the focus is on programming and problem solving, rather than on mathematics. It is our strong conviction that the foundation of computer programming must be carefully laid. Bad habits once begun are hard to change. Even for those who do not continue to study computer science, an experience in the systematic analysis of problems from the statement of "what is to be done" to the final algorithm for "doing it" can be very helpful in encouraging logical thinking.

The programming language presented here is Pascal, a high-level language that encourages good programming style. The language Pascal was devised by Niklaus Wirth and his book "Pascal User Manual and Report" with Kathleen Jensen contains the definition of what is called Standard Pascal. One of the great advantages of Pascal over other high-level languages is that it is not a language with a very large number of language constructs. It is possible because of this to implement it even on very small computer systems from minicomputers to microcomputers. This book can be used with any Pascal compiler that supports Standard Pascal, such as UCSD Pascal and Pascal 6000.

In this book, Standard Pascal is introduced in a series of subsets that we call PS/1, PS/2, PS/3, and so on. The PS stands for Pascal Subsets. The book is about structured programming and that is what we hope a student will be learning by following this step-by-step presentation of Standard Pascal subsets.

Just as a program provides a list of instructions to the computer to achieve some well-defined goal, the methodology of structured programming provides a list of instructions to persons who write programs to achieve well-defined goals. The goals of

structured programming are to get a programming job done correctly and in such a form that later modifications can be done easily. This means that programs must be understood by people other than their authors.

As each Pascal subset is learned, new possibilities open up. Even from the first subset PS/1, it is possible to write programs that do calculations and print. By the time the subset PS/5 is reached, a student has learned how to handle alphabetic information, as well as to do numerical calculations and structure the control flow of the program.

Structured programming is especially important when working on larger programs; a detailed discussion of the techniques of modular programming and top-down design accompanies the introduction of Pascal subprograms in PS/6.

Many examples in the book are from data processing, and in PS/8 the ability to handle files and records is introduced. General concepts of data structures, searching, and sorting fit well into this important area that touches all our lives.

The book includes examples of scientific calculations and numerical methods and a chapter comparing various high-level languages. It ends with a discussion of the operation of a computer and the translation of a high-level programming language into machine language.

At all times we have tried to present things in easy to understand stages, offering a large number of program examples and exercises to be done by the student. Each chapter has a summary of the important concepts introduced in it.

The subsets PS/1, PS/2, PS/3, ..., referred to as a group by the name PS/k, are based on subsets for PL/1 called SP/k designed by Richard Holt and David Wortman of the University of Toronto.

This book was prepared using a text editing system on a computer. Each program was tested using a Pascal compiler. The job of transcribing the authors' pencil scrawls into the computer was done with great care and patience by Inge Weber. The book has been class tested. We are indebted to many people but rather than mentioning a lot of names here we have sprinkled through the book names of people who have helped us.

The time taken to write a book comes at the expense of other activities. Since most of the time was in the evenings or weekends we must end with grateful thanks to our wives Marie and Patricia.

R.C. Holt

J.N.P. Hume

CONTENTS

Chapter 1
INTRODUCTION TO STRUCTURED PROGRAMMING

We hope that it is no secret that the book has to do with computers and particularly with the use of computers rather than their design or construction. To use computers you must learn how to speak their language or a language that they can understand. We do not actually speak to computers yet, although we may some day; we write messages to them. The reason we write these messages is to instruct the computer about some work we would like it to do for us. And that brings us to programming.

WHAT IS PROGRAMMING?

Programming is writing instructions for a computer in a language that it can understand so that it can do something for you. You will be learning to write programs in one particular programming language called Pascal. When these instructions are entered into a computer directly by means of a keyboard input terminal or are put on to some medium that a computer can read such as punched cards and then fed into the machine, they go into the part of the computer called its memory and are recorded there for as long as they are needed. The instructions could then be executed if they were in the language the computer understands directly, the language called machine language. If they are in another language such as Pascal they must first be translated, and a program in machine language compiled from the original or source program. After compilation the program can be executed.

Computers can really only do a very small number of different basic things. For example, an instruction which says, STAND ON YOUR HEAD, will get you nowhere. The repertoire of instructions that any computer understands usually includes the ability to move numbers from one place to another in its memory, to add, subtract, multiply, and divide. They can, in short, do all kinds

of arithmetic calculations and they can do these operations at rates of up to a million a second. Computers are extremely fast calculating machines. But they can do more; they can also handle alphabetic information, both moving it around in their memory and comparing different pieces of information to see if they are the same. To include both numbers and alphabetic information we say that computers are data processors or more generally information processors.

When we write programs we write a sequence of instructions that we want executed one after another. But you can see that the computer could execute our programs very rapidly if each instruction were executed only once. A program of a thousand instructions might take only a thousandth of a second. One of the instructions we can include in our programs is an instruction which causes the use of other instructions to be repeated over and over. In this way the computer is capable of repetitious work; it tirelessly executes the same set of instructions again and again. Naturally the data that it is operating on must change with each repetition or it would accomplish nothing.

Perhaps you have heard also that computers can make decisions. In a sense they can. These so-called decisions are fairly simple. The instructions read something like this:

```
IF JOHN IS OVER 16 THEN PLACE HIM ON THE HOCKEY TEAM
     ELSE PLACE HIM ON THE SOCCER TEAM
```

Depending on the condition of John's age, the computer could place his name on one or other of two different sports teams. It can decide which one if you tell it the decision criterion, in our example being over sixteen or not.

Perhaps these first few hints will give you a clue to what programming is about.

WHAT IS STRUCTURED PROGRAMMING?

Certain phrases get to be popular at certain times; they are fashionable. The phrase, "structured programming" is one that has become fashionable. It is used to describe both a number of techniques for writing programs as well as a more general methodology. Just as programs provide a list of instructions to the computer to achieve some well-defined goal, the methodology of structured programming provides a list of instructions to persons who write programs to achieve some well-defined goals. The goals of structured programming are, first, to get the job done. This deals with how to get the job done and how to get it done correctly. The second goal is concerned with having it done so that other people can see how it is done, both for their education and in case these other people later have to make changes in the original programs.

Computer programs can be very simple and straightforward but many applications require that very large programs be written. The very size of these programs makes them complicated and difficult to understand. But if they are well-structured, then the complexity can be controlled. Controlling complexity can be accomplished in many different ways and all of these are of interest in the cause of structured programming. The fact that structured programming is the "new philosophy" encourages us to keep track of everything that will help us to be better programmers. We will be cataloguing many of the elements of structured programming as we go along, but first we must look at the particular programming language you will learn.

WHAT IS PASCAL?

Pascal is a language that has been developed to be independent of the particular computer on which it is run and oriented to the problems that persons might want done. We say that Pascal is a high-level language because it was designed to be relatively easy to learn. As a problem-oriented language it is concerned with problems of numerical calculations such as occur in scientific and engineering applications as well as with alphabetic information handling required by business and humanities applications.

Pascal is a reasonably extensive language, so that although each part is easy to learn, it requires considerable study to master. Many different computer installations, ranging in size from large computers to microcomputers, have the facilities to accept programs written in Pascal. This means that they have a Pascal compiler that will translate programs written in Pascal into the language of the particular machine that they have. Also many programs have already been written in Pascal; in some installations a standard language is adopted, and Pascal is sometimes that standard language.

It has been the experience over the past years that a high-level language lasts much longer than machine languages, which change every five years or so. This is because once an investment has been made in programs for a range of applications, an installation does not want to have to reprogram when a new computer is acquired. What is needed is a new compiler for the high-level language and all the old programs can be reused.

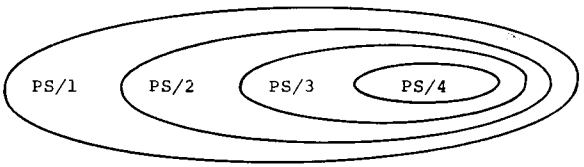
Because of the long life-span of programs in high-level languages it becomes more and more important that they can be adapted to changes in the application rather than completely reconstructed.

A high-level language has the advantage that well-constructed and well-documented programs in the language can be readily modified. Our aim is to teach you how to write such programs.

To start your learning of Pascal we will study subsets of the full Pascal language called PS/k.

WHAT IS PS/k?

The PS in the name PS/k stands for "Pascal Subset". There really is a series of subsets beginning at PS/1, then PS/2, and going on up. The first subset contains a small number of the language features of Pascal, but enough so that you can actually write a complete program and try it out on a computer right away. The next subset, PS/2, contains all of PS/1 as well as some additional features that enlarge your possibilities. Each subset is nested inside the next higher one so that you gradually build a larger and larger vocabulary in the Pascal language. At each stage, as the special features of a new subset are introduced, examples are worked out to explore the increased power that is available.



THE PS/k SUBSETS

In a sense, the step-by-step approach to learning Pascal is structured and reflects the attitude to programming that we hope you learn.

There is no substitute for practice in learning to program, so as soon as possible and as often as possible, submit your knowledge to the test by creating your own programs.

WHY LEARN JUST A SUBSET?

The Pascal language is reasonably extensive; some features are only used rarely or by a few programmers. If you know exactly what you are doing, then these features may provide a faster way to program; otherwise they are better left to the experts. A beginner cannot really use all the features of the complete Pascal language and will get lost in the complexity of the language description. With a small subset it is much easier to pick up the language and then get on with the real job of learning programming.

But perhaps most important, the PS/k language has been selected from the Pascal language so as to provide the basic features that encourage the user to produce well-structured programs. This is why it is so appropriate as a means of learning structured programming.

CORRECTNESS OF PROGRAMS

One of the maddening things about computers is that they do exactly what you tell them to do rather than what you want them to do. To get correct results your program has to be correct. When an answer is printed out by a computer you must know whether or not it is correct. You cannot assume, as people often do, that because it was given by a computer it must be right. It is the right answer for the particular program and data you provided because computers now are really very reliable and rarely make mistakes. But is your program correct? Are your input data correct?

One way of checking whether any particular answer is correct is to get the answer by some other means and compare it with the printed answer. This means that you must work out the answer by hand, perhaps using a hand calculator to help you. When you do work by hand you probably do not concentrate on exactly how you are getting the answer but you know you are correct (assuming you do not make foolish errors). But this seems rather pointless. You wanted the computer to do some work for you to save you the effort and now you must do the work anyway to test whether your computer program is correct. Where is the benefit of all this? The labor saving comes when you get the computer to use your program to work out a similar problem for you. For example, a program to compute telephone bills can be checked for correctness by comparing the results with hand computation for a number of representative customers and then it can be used on millions of others without detailed checking. What we are checking is the method of the calculation.

We must be sure that our representative sample of test cases includes all the various exceptional circumstances that can occur in practice, and this is a great difficulty. Suppose that there were five different things that could be exceptional about a telephone customer. A single customer might have any number of exceptional features simultaneously. So the number of different types of customers might be 32, ranging from those with no exceptional features to those with all five. To test all these combinations takes a lot of time, so usually, we test only a few of the combinations and hope all is well.

Because exhaustive testing of all possible cases to be handled by a program is too large a job, many programs are not thoroughly tested and ultimately give incorrect results when an unusual combination of circumstances is encountered in practice. You must try to test your programs as well as possible and at the

same time realize that with large programs the job becomes very difficult. This has led many computer scientists to advocate the need to prove programs correct by various techniques other than exhaustive testing. These techniques rely partly on reading and studying the program to make sure it directs the computer to do the right calculation. Certainly the well-structured program will be easier to prove correct.

CHAPTER 1 SUMMARY

The purpose of this book is to introduce computer programming. We have begun in this chapter by presenting the following programming terminology.

Program (or computer program) - a list of instructions for a computer to follow. We say the computer "executes" instructions.

Programming - writing instructions telling a computer to perform certain data manipulations.

Programming language - used to direct the computer to do work for us.

Pascal - a popular programming language. PL/1, Fortran, Cobol, Basic and APL are some other popular programming languages.

PS/k - the programming language used in this book. PS/k is a subset of the Pascal programming language, meaning that every PS/k program is also a Pascal program, but some Pascal programs are not PS/k programs. PS/k is itself composed of subsets PS/1, PS/2 and so on. This book teaches PS/1, then PS/2, and so on up to PS/8.

High-level language - a programming language that is designed to be convenient for writing programs. Pascal is a high-level language.

Structured programming - a method of programming that helps us write correct programs that can be understood by others. The PS/k language has been designed to encourage structured programming. This book teaches a structured approach to programming.

Correctness of programs - the validity of programs should be checked. This can be attempted by comparing test results produced by the computer with the results of calculations made in another way, e.g., by using a hand calculator. Although the ideal is to try to prove a program correct by mathematical means, it is often extremely helpful to read and study the program to see that your intentions will be carried out.