PASCAL
An Introduction to Methodical Programming

W Findlay & D A Watt

Pascal

An Introduction to Methodical Programming

William Findlay & David A. Watt
Computing Science Department,
University of Glasgow

Pitman

Pitman Publishing Limited
39 Parker Street London WC2B 5PB

Associated Companies
Copp Clark Pitman Toronto
Fearon Pitman Publishers Inc San Francisco
Pitman Publishing New Zealand Ltd Wellington
Pitman Publishing Pty Ltd Melbourne

First Published in Great Britain 1978

Reprinted 1979

Printed by photolithography and bound in Great Britain
by Biddles Ltd Guildford Surrey

ISBN 0 273 01220 7

Preface

This book is intended for use in conjunction with a first course in computer programming based on the programming language Pascal. The reader is assumed to have had no previous exposure to computers, and to have only elementary mathematics. Programming principles, good style and a methodical approach to program development are emphasized, with the intention that the book should be useful even to those who must later write programs in a language other than Pascal. Thus our primary objective is simply to teach readers how to write good programs.

A secondary objective is to present an introduction to Pascal. In this respect the book should be useful not only to novices but also to readers with some limited experience of programming in another language.

Pascal was introduced in 1971 by Professor Niklaus Wirth. His aim was to make available a language which would allow programming to be taught as a systematic discipline and in which the techniques of both "scientific" and "commercial" programming could be convincingly demonstrated. The adoption of Pascal has been rapid and widespread, to the extent that it has become the <u>lingua franca</u> of computing science.

For our present purposes what is really important is the clarity with which fundamental programming concepts may be expressed in Pascal. Most of the book is devoted to a treatment of these fundamentals, presented in such a way that the reader should be convinced of the <u>need</u> for each language feature before he is shown how it is realized in Pascal. Since Pascal contains only a few features which are not truly fundamental, these remaining features are also covered, briefly, for the sake of completeness.

Use of the book

The best way to acquire a methodical approach to programming is subconsciously, by imitation, and the best time to start is right at the beginning. The technique of programming by stepwise refinement is therefore imparted mainly by consistent example throughout the book. Nevertheless, two chapters are devoted exclusively to programming methodology. The first, Chapter 7, introduces the methodology by means of a case study, and is placed early enough to encourage good programming habits from the start. The second, Chapter 20, applies the methodology to realistically-sized problems, by means of two further case studies. Although this chapter comes at the end of the book, the case studies can and should be read at an earlier stage: Case Study II after Chapter 16, and Case Study III after Chapter 17.

The main text falls naturally into six parts. Part I (First Steps in Programming) aims to bring the novice as soon as possible to the stage of writing and testing complete programs in a methodical manner. This part covers the INTEGER and BOOLEAN data types, input and output, and the basic control structures of sequencing, selection and repetition. Its highlights are the first complete program, in Chapter 4, and the introduction of a methodology, in Chapter 7. Part II (More Data Types) covers the remaining simple data types, such as CHAR and REAL, and arrays. Part III (More Control Structures) completes the treatment of control structures. Part IV (Subprograms) introduces functions and procedures. This is the pivot of the book - the reader who has mastered the material up to this point can reasonably call himself a programmer. Part V (More Data Structures) completes the coverage of Pascal's rich variety of data structures with records, strings, files, sets and pointers. Most of these features are not found in many other programming languages, but they contribute substantially to Pascal's expressive power. Part VI (Programming Methodology) consists of the chapter of case studies.

Some of the topics could be skipped on a first reading, and are so marked in the list of contents and in the text. These same topics may be omitted altogether if time presses.

Examples

Every non-trivial example used in this book has been tested on a computer. We challenge readers to find any errors in them!

Exercises

Each chapter is followed by a set of exercises. The more difficult exercises are marked with asterisks (*). Some of the exercises are intended to be answered on paper, to provide practice in the use of the language features introduced in the chapter. Answers to a selection of such exercises are provided. The remaining exercises are designated programming exercises, which involve the writing of complete programs to be be run and tested on a computer. Practical experience of this nature is essential to every programmer. (Not all the programming exercises need be attempted.)

A programming course should be supplemented by a programming laboratory, in which a series of programming exercises selected by the course organizer should be undertaken. The programming exercises herein may be used to assist in such a selection.

References

We have not attempted to write a work of reference, but we hope that the arrangement of the material, together with the appendices and the index, will assist the reader to find information on specific points. The standard reference on Pascal is "Pascal User Manual and Report" by Kathleen Jensen and Niklaus Wirth (Springer-Verlag, New York-Heidelberg-Berlin, 1975). For those wishing to study programming further we can wholeheartedly recommend "Algorithms + Data Structures = Programs" by Niklaus Wirth (Prentice-Hall, Englewood Cliffs, New Jersey, 1976).

Acknowledgments

Like all programmers, we owe a great debt to Professor Edsger Dijkstra, whose insights into the creative aspects of programming we have attempted to reflect. Equally, we wish to acknowledge the work of Professor Niklaus Wirth, whose programming language Pascal is by far the best tool available today for teaching the fundamental concepts of programming.

We also wish to thank our colleagues in the Computing Science Department of Glasgow University for their encouragement and advice, and in particular Dr John Jeacocke whose perceptive comments were of great assistance. Our gratitude goes to Professor D.C. Gilles for allowing us ready access to the Department's PDP 11/40 computer, which we used to prepare and type camera-ready copy.

W. Findlay
D. A. Watt

April 1978

Contents

{Topics marked with an asterisk may be omitted on a first reading.}

PREFACE v

PART I FIRST STEPS IN PROGRAMMING

Chapter 1. Computers and programming 1
1.1. Introduction 1
1.2. Hardware and software 2
1.3. Programming languages 3
Exercises 1 5

Chapter 2. Data: types, constants and variables 8
2.1. Data and data types 8
2.2. Constants and variables 9
2.3. Definitions and declarations in Pascal 10
2.4. Assignment statements 12

Chapter 3. The data type INTEGER 15
3.1. INTEGER constants and variables 15
3.2. INTEGER expressions and assignments 16
Exercises 3 17

Chapter 4. Input and output; the complete program 18
4.1. Input and output of data 18
4.2. Basic input 19
4.3. Basic output 20
4.4. The complete program 21
4.5. Programming style 24
4.6. Further input and output 25
Exercises 4 28

Chapter 5. The data type BOOLEAN 30
5.1. Conditions in programming 30
5.2. Comparisons 31
5.3. The BOOLEAN operators 32
5.4. Using BOOLEAN data 34
5.5. *The rules of Boolean Algebra 35
Exercises 5 37

Chapter 6. The flow of control 39
6.1. Control structures 39
6.2. The WHILE statement 40
6.3. The IF statement 45
6.4. Nested logic 47
Exercises 6 49

Chapter 7. Programming methodically 50
7.1. Programming by stepwise refinement 50
7.2. Making refinements 55
7.3. Testing and correcting 57
7.4. Documentation 59
7.5. Case Study I: the traffic survey problem 60
7.6. The software development cycle 66
Exercises 7 67

PART II MORE DATA TYPES

Chapter 8. Ordinal types and type definitions 69
8.1. The data type CHAR 69
8.2. Enumeration types 72
8.3. Subrange types 75
8.4. Ordinal types in general 77
8.5. Type definitions 80
Exercises 8 81

Chapter 9. The data type REAL 83
9.1. Real numbers and real arithmetic 83
9.2. Real arithmetic in Pascal 85
9.3. Programming with REAL data 88
Exercises 9 91

Chapter 10. Arrays 93
10.1. The need for arrays 93
10.2. Arrays in Pascal 94
10.3. Elementwise processing of arrays 98
10.4. Types of subscripts and elements 101
10.5. Multi-dimensional arrays 104
Exercises 10 106

PART III MORE CONTROL STRUCTURES

Chapter 11. More about flow of control 109
11.1. What further control structures do we need? 109
11.2. Multi-way selection: the CASE statement 109
11.3. Count-controlled loops: the FOR statement 113
11.4. *Loops with at least one iteration: the REPEAT statement 118
11.5. *Exceptional flow of control: the GOTO statement 121
Exercises 11 124

PART IV SUBPROGRAMS

Chapter 12. Functions 127
12.1. The need for functions 127
12.2. Functions in Pascal 127
12.3. Declarations inside functions 132
Exercises 12 135

Chapter 13. Procedures 137
13.1. The need for procedures 137
13.2. Procedures in Pascal 138
13.3. Value-parameters and variable-parameters 141
13.4. The virtues of procedures and functions 149
13.5. The scope rules of Pascal 153
Exercises 13 158

Chapter 14. *Advanced use of functions and procedures 160
14.1. *Recursive functions and procedures 160
14.2. *Functional-parameters and procedural-parameters 165
Exercises 14 167

PART V MORE DATA STRUCTURES

Chapter 15. Records 169
15.1. The need for records 169
15.2. Records in Pascal 170
15.3. Operations on complete records 173
15.4. The WITH statement 175
15.5. Data structures 177
15.6. *Variant records 180
Exercises 15 184

Chapter 16. Packed data and strings 186
16.1. Packed data 186
16.2. The PACKED attribute in Pascal 187
16.3. Strings and string handling 188
Exercises 16 192

Chapter 17. Files 195
17.1. Files and file structures 195
17.2. Files in Pascal 197
17.3. Text files 201
17.4. *File buffers 204
Exercises 17 207

Chapter 18. *Sets 209
18.1. *Sets and their use in programming 209
18.2. *Sets in Pascal 210
Exercises 18 215

Chapter 19. *Pointers and linked lists 216
19.1. *Pointers 216
19.2. *Dynamic data structures 217
19.3. *Pointers in Pascal 218
19.4. *Linked list processing 220
Exercises 19 224

PART VI PROGRAMMING METHODOLOGY

Chapter 20. Programming methodology: case studies 225
20.1. Programming by stepwise refinement: a review 225
20.2. Case Study II: text formatting 226
20.3. Case Study III: transferable vote elections 244
20.4. Some general principles 257
Exercises 20 262

APPENDICES

Appendix 1. Collected syntax diagrams 265
1.1. The program 265
1.2. Statements 266
1.3. Expressions 268
1.4. Variables 269
1.5. Definitions and declarations 270
1.6. Types 272
1.7. Constants 273
1.8. Identifiers 274

Appendix 2. Reserved words and special symbols 275
2.1. Reserved words 275
2.2. Special symbols and their representations 275

Appendix 3. Predeclared entities 276

Appendix 4. Legible input and output 280
4.1. The procedure READ 280
4.2. The procedure READLN 280
4.3. The procedure WRITE 281
4.4. The procedure WRITELN 281
4.5. Other procedures and functions 281

Appendix 5. Character sets 282
5.1. The ASCII character set 282
5.2. The EBCDIC character set 282

ANSWERS TO SELECTED EXERCISES 283

INDEX 303

Part I First Steps in Programming

Chapter 1 Computers and programming

1.1 INTRODUCTION

There can be few people, at least in the industrialized countries of the world, who have never had any contact with computers. Computers are now routinely used for mundane tasks such as producing bank statements, financial reports, electricity bills and payslips. Hotel and airline reservation systems have been made possible by computers. In industry, computers control machine tools and chemical plant. Scientists use computers to analyse experimental data, doctors generate "cross-section" X-ray pictures, psychologists simulate mental processes. Manned and unmanned space exploration would hardly be possible without the assistance of computers. On the frivolous side, computers have been programmed to play games such as backgammon and chess (but not very well). More ominously, military applications have a long history.

Computing has grown from nothing, just thirty years ago, to a position as one of the world's largest industries. There no sign that this expansion is slowing down. Indeed the development of cheap integrated circuits means that domestic and personal computers are now becoming practical. These will more and more invade everyday life as domestic appliances, motor vehicles, communications systems and the like come increasingly to depend on them. This accelerating process has rightly been called the Second Industrial Revolution. Nobody can yet foresee with any certainty what the ultimate consequences for society may be, but it is already clear that vast changes lie ahead for us all.

Consider the impact of personal calculators on accepted ideas about education and numeracy. Computer technology will soon have a similar effect in all areas of clerical and skilled manual work. This book was prepared using a computer, making it considerably less expensive than would be possible with traditional printing technology. On the other hand the craft of the compositor has been made redundant, and the end product lacks the elegance he might have given it. Concerns like these make it imperative that computers be understood as widely as possible.

One of the most common misconceptions is that computers are "problem-solving" machines. Nothing could be further from the truth. In fact the successful application of computers is made possible only by finding solutions to problems which computers themselves have created. The most obvious of these is that a computer is useless without a program to control it. The writing of good computer programs is both a vital part of the modern economy and a fascinating intellectual exercise. Such is the topic of this book.

1.2 HARDWARE AND SOFTWARE

Early computers filled large rooms with tall metal racks on which were fixed thousands of vacuum tubes, tanks of hot mercury and panels of flashing lights. The resemblance to an ironmonger's store was so compelling that the computer engineers of the time wryly talked about their creations as "hardware". Nowadays a considerably more powerful computer fits easily in a briefcase, but the principles of its operation are the same.

The hardware of every digital computer consists of a processor, a store and an assortment of peripheral devices. The processor is the unit which actually performs the calculations. It contains a control unit to direct operations, as well as an arithmetic unit. The latter is equivalent to an electronic calculator, but much faster, being capable of a million or more operations per second. To make use of this speed the processor must be able to access its data equally quickly. Retaining data for rapid access by the processor is the job of the computer's store. Some calculators have a handful of "registers" in which numbers can be kept. The store of a modest computer contains tens of thousands of registers. A calculator's numeric keys and display correspond to the peripherals or input/output devices of a computer. These allow data to be placed in the store and results to be taken out. Though very fast by human standards, peripherals are usually much slower than the processor and store.

A calculator is given instructions by pressing its function keys. However the great speed of a computer would be wasted if it could not be supplied with instructions as quickly as it obeys them. To make this possible the computer's instructions, encoded in numerical form, are held in store along with the data. The computer works in a cycle as follows.

(1) The control unit fetches the next instruction from store.
(2) The instruction is decoded into electronic signals by the control unit.
(3) In response to these signals the arithmetic unit, the store, or a peripheral device carries out the instruction.
(4) The whole cycle repeats from step (1).

In this way long sequences of instructions can be obeyed automatically at the full speed of the processor. Such a sequence of instructions is called a program.

Computer instructions are very simple in their effect, the following examples being typical.
(a) Read an item of data into store from an input device.
(b) Copy an item of data from one register to another.
(c) Add the contents of two registers and place the sum in a third.
(d) If the content of a register represents a negative number, take the next instruction from a different part of the program; otherwise continue with the next instruction in sequence.
(e) Write an item of data from store to an output device.

It has been proved that anything which can be computed, in principle, can be computed in a finite number of steps by a program consisting of elementary operations such as these. Such a program is called an algorithm. It has also been proved that there are results which are not computable by any machine whatsoever. In these cases it may be possible to compute an approximation to the desired result. A program to do this is called a heuristic. Heuristics are also useful when an algorithm exists but is impractically slow or needs too much store.

The collection of all the programs available in a computer system constitutes its software. This word was invented to emphasize that the programs are just as important as the hardware. It also contrasts them effectively. Hardware is visible, solid and substantial; software is somewhat intangible. The hardware of a computer system is not easily changed; the software is usually in a state of flux.

One of the most important parts of the software is the operating system, a set of control programs which are kept permanently in store. The operating system carries out many of the routine tasks needed to prepare and run a user's job, e.g. deciding which job to run next, making ready its input, bringing the user's program into store, allocating it some processor time, and so on.

1.3 PROGRAMMING LANGUAGES

The earliest computers were programmed in machine code: i.e., by giving them instructions directly in numerical form. However the drawbacks were soon recognized.

(a) Because of the very primitive nature of machine instructions, machine-code programming is both tedious and error-prone.

(b) For the same reason, machine-code programs are difficult to understand and to modify.

(c) Programming is a time-consuming and expensive business. It would be a great saving to be able to transfer programs between computers, but a machine-code program is specific to one model of computer and will not work on any other.

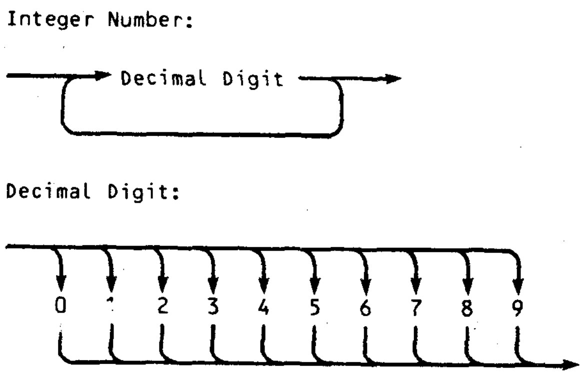
Why not write programs in English? Computing is not unique in requiring the detailed description of sequences of actions: there are many similar examples in daily life. However, anyone who has ever struggled with the often mystifying instructions in motor maintenance handbooks, do-it-yourself manuals, or recipe books will readily agree that English is far from ideal for the job. In fact the glories of English - its vast scope, its subtlety, its potential for ambiguity and metaphor - must be considered severe disadvantages when the aim is literalness, accuracy and completeness. A programming language must aim at the truth, the whole truth, and nothing but the truth.

English is at the opposite extreme from machine code and precisely for that reason must be rejected as a medium for practical computer programming. What is needed is a middle way: one which combines the readability and generality of English with the directness and precision of machine code. Because of their position relative to machine code, languages of this sort are called high-level languages.

Knitting patterns offer an interesting example where a similar problem has been faced. A knitting pattern is comparable in complexity with a modest computer program, so it is understandable that a special "knitting language" has evolved. It borrows many words from English, but these are used in stereotyped ways and with definite meanings. Another noteworthy feature of a knitting pattern is its division into two parts: a list of the materials and tools needed, followed by a list of instructions stating how to use them. The programming language used in this book, Pascal, shares both of these characteristics.

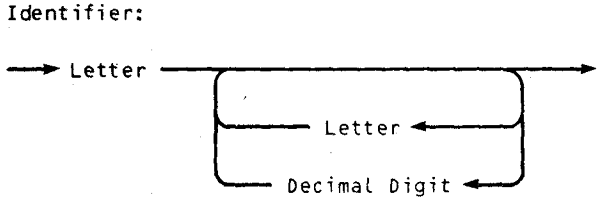
Any language can be studied from two points of view: that of its grammar, or syntax, and that of its meaning, or semantics. A good understanding of both is needed to use it properly. We will find that the semantics of Pascal can be described adequately in English. On the other hand a description of its syntax in English would be very tedious. Instead we will use a pictorial device, the syntax diagram. This is best explained by an example. Stated in English, the Pascal definition of an Integer Number is the following. "An Integer Number is a sequence of one or more Decimal Digits. A Decimal Digit is the character '0', or '1', or '2', or '3', or '4', or '5', or '6', or '7', or '8', or '9'." Exactly the same information is conveyed by Figure 1.1.

Figure 1.1. Syntax of Integer Numbers



This can be understood by following the arrows through the diagram, from entry to exit, and writing down a specimen of everything you pass. When you come to a fork, either path may be chosen. As a slightly more complicated example, Figure 1.2 defines an Identifier, widely used in Pascal to give things names. The English equivalent of this diagram is "An Identifier is a sequence of characters beginning with a Letter and followed by zero or more Letters or Decimal Digits." The definition of Letter is omitted: it is tedious in any form.

Figure 1.2. Syntax of Identifiers



A program written in Pascal cannot be directly performed by the hardware of a computer. To make it executable it must be translated from Pascal into an equivalent set of machine code instructions. This translation can be specified rigorously enough to make it a suitable task for a computer program. Three programs are involved here: the translator program, or compiler; the user's Pascal text, or source program; and the equivalent machine code, or object program. Thus a Pascal program is run in two distinct stages.

(1) The Pascal compiler is brought into store and activated. It causes the computer to read the source program, check it for errors, and convert it into the corresponding object program.
(2) The object program is left in store as the result of stage (1). It is activated in turn and reads input, performs computations, and writes output in exactly the manner described by the original Pascal program.

Programs often contain errors in the use of the programming language and these are reported by the compiler during stage (1). The report usually takes the form of an error message or a number which refers to a list of error messages. These error messages are often helpful in finding the cause of the trouble. (However, the compiler may be misled by an error into taking later, perfectly correct, parts of a program as erroneous. Thus one genuine error can cause a whole group of messages to be output, many of which are spurious. Nothing more forcefully reminds a programmer that computers do not understand the programs which drive them.)

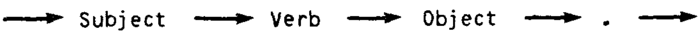
EXERCISES 1

1.1. Which of the following are valid Identifiers, according to the syntax of Figure 1.2?

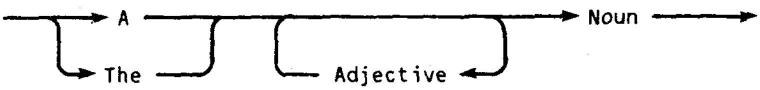
(a) SEVEN, (b) VII, (c) NMR7, (d) 7, (e) KERMIT, (f) JOE 90,
(g) ABCDEFGHIJKLMNOPQRSTUVWXYZ

1.2. Write down examples of Sentences generated by the following syntax diagrams.

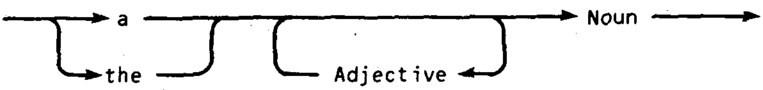
Sentence:



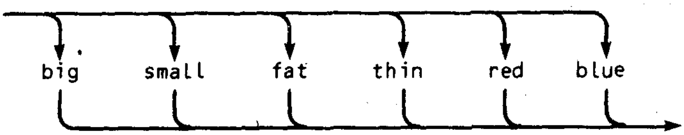
Subject:



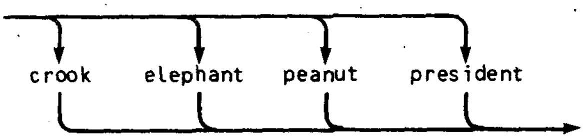
Object:



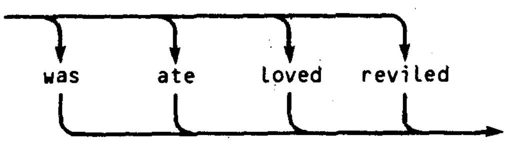
Adjective:



Noun:



Verb:



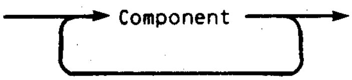
1.3. Which of the following are valid Sentences, according to the syntax of the previous exercise?

(a) The big fat president was a crook.
(b) The reds reviled the president.
(c) The elephant ate peanuts.

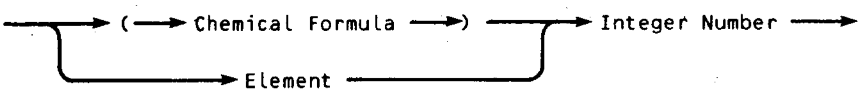
(d) The small, thin president loved a peanut.
(e) The thin crook was a red.
(f) A elephant was a elephant.

1.4. Write down examples of Chemical Formulas generated by the following syntax diagrams.

Chemical Formula:



Component:



Element: