Jim Welsh &
John Elder

Introduction to Pascal

ENTICE-HALL
TERNATIONAL
SERIES IN
COMPUTER
SCIENCE

C. A. R. HOARE SERIES EDITOR

INTRODUCTION TO PASCAL

by

JIM WELSH

and

JOHN ELDER

Queen's University of Belfast
Northern Ireland

ENGLEWOOD CLIFFS, N.J. LONDON NEW DELHI SINGAPORE
SYDNEY TOKYO TORONTO WELLINGTON

Library of Congress Cataloging in Publication Data

WELSH, JIM 1943–
Introduction to Pascal.

Includes index.
1. PASCAL (Computer program language)
I. Elder, John, 1949– joint author.
II. Title.
QA76.73.P35W44 001.6'424 78–27419
ISBN 0–13–491522–4

British Library Cataloguing in Publication Data

WELSH, JIM
Introduction to Pascal.
1. PASCAL (Computer program language)
I. Title II. Elder, John.
001.6'424 QA76.73.P35
ISBN 0–13–491522–4

ISBN 0–13–491522–4

PRENTICE-HALL INTERNATIONAL, INC., *London*
PRENTICE-HALL OF AUSTRALIA PTY. LTD., *Sydney*
PRENTICE-HALL OF CANADA, LTD., *Toronto*
PRENTICE-HALL OF INDIA PRIVATE LIMITED, *New Dehli*
PRENTICE-HALL OF JAPAN, INC., *Tokyo*
PRENTICE-HALL OF SOUTHEAST ASIA PTE., LTD., *Singapore*
PRENTICE-HALL, INC., *Englewood Cliffs, New Jersey*
WHITEHALL BOOKS LIMITED, *Wellington, New Zealand*

Printed in the United States of America

10 9 8 7 6 5 4 3 2

INTRODUCTION TO PASCAL

Prentice-Hall International
Series in Computer Science

C.A.R. Hoare, Series Editor

Published

BACKHOUSE, R. C., *Syntax of Programming Languages: Theory and Practice*
DUNCAN, F., *Microprocessor Programming and Software Development*
WELSH, J., and ELDER, J., *Introduction to PASCAL*

Future Titles

HENDERSON, P., *Functional Programming*
JACKSON, M. A., *System Design*
JONES, C., *Software Development*
WELSH, J., and MCKEAG, M., *Structured System Programming*

Preface

The programming language PASCAL was developed in the late 1960s by Professor Niklaus Wirth at the Eidgenössische Technische Hochschule, Zürich, Switzerland. His aim was to produce a language containing a small number of fundamental programming concepts that would be suitable for teaching programming as a logical and systematic discipline, and also be capable of efficient implementation on most computers. His success in achieving this goal can be measured by the rapid and widespread increase in the use of PASCAL, both as a language for teaching the principles of computer programming and as a practical language for writing systems and applications programs.

This book provides a comprehensive introduction to PASCAL and is suitable for use by novice programmers and by those with a knowledge of other programming languages. The complete language is described and the use of all the features of PASCAL is fully illustrated. The language which we describe does not contain any features peculiar to any particular implementation—indeed great care is taken to isolate and stress the implementation-dependent features of PASCAL. More general principles of programming are illustrated implicitly in the text. In fact, the style of programming used is consistent with the current methods of structured programming and stepwise refinement.

The material is based on various courses given at The Queen's University of Belfast over the past seven years by the authors, who have wide experience in the teaching, use, and implementation of PASCAL. The teaching sequence of the book is that used on those courses and is suitable for a reader learning PASCAL, or indeed learning to program, for the first time. The other aspects of computers and computer organization of which a first-time programmer must be aware are not covered in detail, but Chap. 1 presents a summary of the

knowledge and terminology on which subsequent chapters depend. Chapters 3–6 introduce the basic data types and statements of PASCAL. Chapter 7 deals with procedures and functions, the use of which we consider fundamental in the program-construction process. In Chaps. 9–13 PASCAL's data-structuring facilities—arrays, strings, records, sets, files and pointers—are introduced.

The features of PASCAL are presented in turn, by

(a) defining the feature using the notation and terminology of the standard PASCAL definition;
(b) explaining its use, and any limitations and practical problems of which the user must be aware;
(c) illustrating its use by small, tested, program fragments imbedded within the explanatory text.

A distinguishing feature of the book is the inclusion, where appropriate, in each chapter of one or more complete case-study programs. In all the book contains seventeen such case studies illustrating the use of various PASCAL facilities, and basic computing algorithms, in a significant practical context. For each case study the design of the program is developed by means of a step-wise refinement of the problem, and a final listing of the PASCAL program and the results it produces is reproduced directly from its computer-printed output.

Most chapters end with a set of programming exercises which involve further use of the language features described in the chapter. These exercises require modifications or extensions to be made to earlier case-study programs as well as the construction of new programs.

The book is intended for use in both learning and reference mode. To facilitate its use in reference mode the book ends with an Appendix of syntax diagrams, and a comprehensive index. The syntax diagrams provide a concise summary of the language features in a form which is easily used by someone familiar with the basic framework of the language. The index lists all formal and informal terms used in the text showing the defining occurrence of each, together with other occurrences which may help to clarify its significance.

At the time of writing, the universally accepted standard definition of PASCAL is that given in Jensen and Wirth's book *Pascal User Manual and Report*, published by Springer–Verlag. Throughout this text the terms "the definition of PASCAL" and "the PASCAL Report" are used to refer to this standard.

Belfast, January 1979

JIM WELSH
JOHN ELDER

ACKNOWLEDGEMENTS

We wish to thank Professor Tony Hoare, who encouraged us to write the book, and all those people who read drafts of the manuscript, pointed out mistakes to us, and made constructive suggestions.

Contents

1

Computers and Programming

Computer programming requires an understanding of the nature of computers, of computer programs and of the programming languages in which programs may be expressed. Subsequent chapters of this book explain how computer programs may be written in the programming language PASCAL. This first chapter summarizes the general concepts of computers and their programming on which the subsequent chapters depend.

For those readers who have already programmed computers in other languages the chapter may provide a summary of the terminology, and perhaps a hint of the programming philosophy, used in the following chapters.

For those learning to program for the first time this chapter provides a very brief summary of the facts and concepts which they must come to appreciate while programming in PASCAL or any other language. A course instructor may provide a more detailed treatment of these topics before or during study of the material covered in the following chapters.

THE COMPUTER

A *computer* is a machine which can carry out long, complex and repetitive sequences of operations at very high speed. These operations are applied to *information* or *data* supplied by the user to produce further information or *results* which the user requires. The sequence of operations required to produce the desired results in any particular computing task is specified as a *computer program* prepared for that task.

The essential components of a computer are a *processor*, a *memory*, and some *input and output devices.*

The *processor* is the work horse which carries out the sequence of operations specified by the program. The individual operations provided by the processor are very simple but are carried out at very high speed—perhaps one million or more operations per second.

The *memory* is used to store the information to which the processor's operations are applied. Memory is of two kinds—*primary* or *main store*, and *secondary* or *backing store.* The main store enables the processor to fetch and to store units of information at a speed which is comparable to its speed of operation, in fact each operation normally involves at least one store access. To enable the processor to proceed from one operation to the next without delay the sequence of program instructions which specifies these operations is also held in the main store. The main store thus holds both *instructions* and *data*, on which the processor operates.

The amount of main store available to the processor is limited, and is used to hold only the programs and data on which the processor is operating at that time. It is not used for any permanent storage of data. In some cases the main store may not even be large enough to hold all the instructions or data involved in the execution of one program. For these reasons computers are also equipped with secondary storage devices such as magnetic tapes, disks or drums. The essential characteristics of these devices are:

(a) their capacity is normally much greater than that of the main store;
(b) information can be held by them permanently, e.g. from one program execution to another.

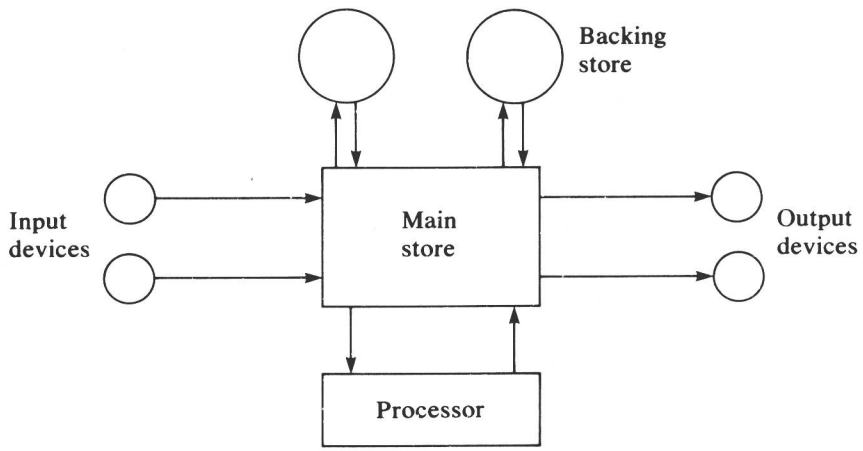

Fig. 1.1 Organization of a computer.

However, the speed at which information can be transferred to and from secondary storage is much lower than for main store.

Input and output devices are used to transfer information from the outside world to the computer's main store (*input*) and from the main store to the outside world (*output*). Familiar input devices are card readers, which read information punched on cards into the computer's store, and terminal keyboards which transfer the characters indicated by key depressions into the computer store. Familiar output devices are line printers and typewriters which print information on continuous paper, and visual display screens which display textual or graphic information on the screen of a cathode ray tube.

The logical organization of a computer may thus be depicted as in Fig. 1.1.

WRITING A COMPUTER PROGRAM

The use of a computer for a particular task involves three essential steps:

(a) specifying the task the computer is to carry out, in terms of the input data to be supplied and the output data or results to be produced;
(b) devising an *algorithm* or sequence of steps, by which the computer can produce the required output from the available input;
(c) expressing this algorithm as a computer program in a programming language such as PASCAL.

Step (a), the specification, is not normally considered as part of the programming process but a precise specification is an essential prerequisite for a successful program.

It has been common practice in the past to separate steps (b) and (c), first defining the algorithm in a notation convenient for its design, and then translating or encoding this design into the chosen programming language. However, the language PASCAL provides a notation which may be used both for the design and for the final coding of the program required. With PASCAL, therefore, steps (b) and (c) are not usually separated, but merged as a continuous design/programming process. This approach is well illustrated by the case-study programs considered in the following chapters of this book.

In principle, once the computer program has been written, the programmer's task is complete, since execution of this program by the computer should produce the required results. In practice, because the task to be carried out by the computer is complex, and the human

programmer's ability is limited, the first program written may not produce the required results. The programmer, therefore, engages in a cycle of checking and correcting his program until he is satisfied that it meets its specification completely. This process of detecting and correcting errors in a program is known as *debugging*. Debugging is commonly accomplished by running the program on the computer with suitable test data.

RUNNING A COMPUTER PROGRAM

In a "high-level" language such as PASCAL the program is expressed as a sequence of elementary steps which are convenient to the programmer. Likewise the program is prepared in a form which is convenient for the programmer to generate—as a piece of text written or printed on paper, punched on a sequence of cards, or typed at a computer-terminal keyboard.

However, the program which the computer's processor executes must be expressed as a sequence of the much simpler "low-level" operations available to the processor, and must be held in the computer store as a sequence of encoded instructions each of which is immediately executable by the processor. Preparation of a program in this form is an extremely tedious and painstaking task for a human programmer.

Fortunately, however, the translation of a program text expressed in a high-level language into an equivalent sequence of processor-executable instructions within the computer store is itself a routine task which can be carried out by a computer program. Such a program is provided for each high-level language which may be used on a computer, and is known as the *compiler* for that language.

Thus a program written in a high-level language in text form is first input to the computer as data for an execution of the compiler program for that language. The compiler produces an equivalent executable program in the computer store, which may then be executed or run to produce the desired effect. Figure 1.2 shows this two-stage process in schematic form.

In translating a high-level language program the compiler may detect many of the simple mistakes which the programmer has made in expressing his program in that language. The compiler reports these errors to the programmer by outputting a *program listing*, i.e., a printout of the original program input together with messages identifying any error detected during its compilation. Program errors detected in this way are known as *compile-time errors*.

In running the executable program produced by the compiler, further