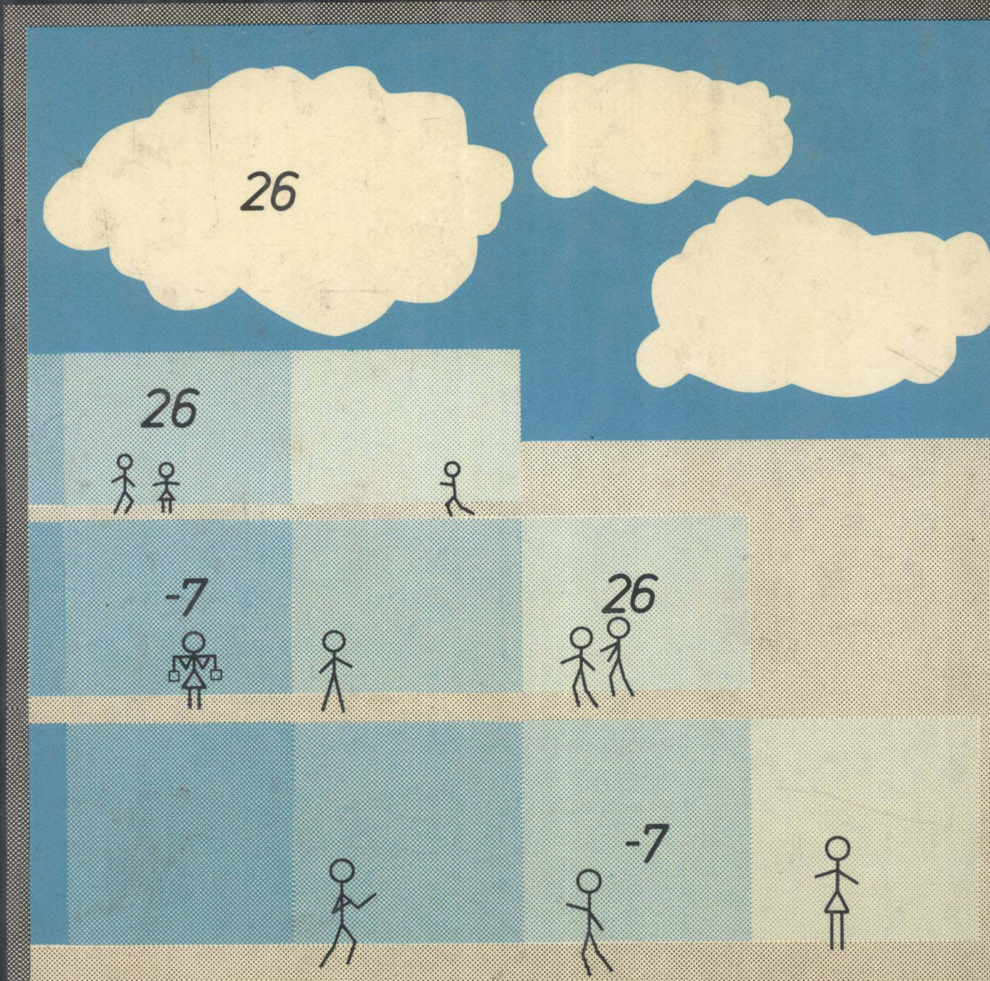
Ellis Horwood · Publishers
COMPUTERS AND THEIR APPLICATIONS 2

Introductory Algol 68 Programming

D.F. Brailsford A.N. Walker



INTRODUCTORY
ALGOL 68
PROGRAMMING

INTRODUCTORY ALGOL 68 PROGRAMMING

D. F. BRAILSFORD and A. N. WALKER
Department of Mathematics,
University of Nottingham

ELLIS HORWOOD LIMITED
Publishers Chichester

Halsted Press. a division of
JOHN WILEY & SONS
New York - Chichester - Brisbane - Toronto

First published in 1979 by

ELLIS HORWOOD LIMITED
Market Cross House, Cooper Street, Chichester, West Sussex, PO19 1EB, England

The publisher's colophon is reproduced from James Gillison's drawing of the ancient Market Cross, Chichester

Distributors:

Australia, New Zealand, South-east Asia:
Jacaranda-Wiley Ltd., Jacaranda Press,
JOHN WILEY & SONS INC.,
G.P.O. Box 859, Brisbane, Queensland 40001, Australia.

Canada:
JOHN WILEY & SONS CANADA LIMITED
22 Worcester Road, Rexdale, Ontario, Canada.

Europe, Africa:
JOHN WILEY & SONS LIMITED
Baffins Lane, Chichester, West Sussex, England.

North and South America and the rest of the world:
HALSTED PRESS, a division of
JOHN WILEY & SONS
605 Third Avenue, New York, N.Y. 10016, U.S.A.

British Library Cataloguing in Publication Data

Brailsford, D. F.
Introductory Algol 68 programming. –
(Ellis Horwood series in computer science).
1. ALGOL (Computer program language)
I. Title II. Walker, A. N.
001.6'424 QA76.73.A24 79–40241

ISBN 0–85312–127–3 (Ellis Horwood Ltd., Publishers)
ISBN 0470–26746–1 (Halsted Press)

Printed and bound in Great Britain by
Fakenham Press Ltd, Fakenham, Norfolk

Table of Contents

Foreword

This book is intended for anyone who wants to learn to write programs in Algol 68. It is based on the course which we have given for some years at Nottingham to mathematics undergraduates learning their first programming language. (It would perhaps be more accurate to say that it is based on the course we would have given had a suitable textbook been available!) We certainly do not require you to be an expert mathematician – the mathematical content of this book is minute. Nor do we require you to know some other programming language already; if you do, you may be able to skim some parts rather quickly, but you will also probably have some bad habits to unlearn (as we did, when we came to Algol 68). We do assume a little basic knowledge of computer appreciation, as outlined in the Introduction; a logical mind is desirable, access to a computer is probably essential, and access to an Algol 68 expert will be helpful.

In its formative years, Algol 68 had a very bad press. Stories circulated of formidably obscure documents and of the high priests of the language deliberating in conclave on whether a particular full stop in the defining report ought to be in italics. As usual, there was a grain of truth in these stories, and a mountain of exaggeration; new initiates may rest assured that there is nothing in the use of Algol 68 (as possibly opposed to its rigorous definition) which is in any way harder than the corresponding use of Fortran (or Algol 60, Basic, Pascal, PL/1, APL or any other computing language). We are certainly not high priests of Algol 68 (not even priests – perhaps missionaries). Our concern is to write programs in the most effective way possible, and to teach others to do so; and we are quite sure that Algol 68 is currently far and away the best vehicle for writing general purpose programs simply, efficiently and correctly. We hope we can convince you.

We have had to steer a careful course between Algol 68R – the first practical and working version of the language – and Revised Algol 68, for which effective compilers have – at last! – recently appeared (for historical details, see the Introduction and Appendix 7). When we wrote the first draft of this book, in 1972, we concentrated on 68R. We had ourselves seen the output from some

20000 (mainly student) programs in that dialect, certainly more than half the world total, whereas the real language existed only on paper. Today, Algol 68R still accounts for perhaps 90% of the world total throughput of Algol 68 programs – its extensive use in commerce, industry and (applied) research ensures that – but educational practice is slowly but surely moving towards the purer form of the language. Revised Algol 68 is available on every important model of computer, and is absolutely standard and understood everywhere, whereas 68R is specific to large ICL machines. Our revised text therefore treats 68R as the heresy – albeit a very important one. We have tried to discuss topics in a way which applies to both versions, at the occasional expense of generality.

A note for the untutored reader. We have concentrated on the elementary parts of the language because there is adequate documentation in the existing literature of the advanced parts, because the further developments depend significantly on your interests and on the equipment available for your use, and because the elementary bits constitute on their own a self-contained programming language which is simpler to use, generally more efficient, and appreciably more powerful than traditional languages such as Fortran and Algol 60. Nevertheless, you may find some (a little, we hope) of the material in this book hard and subtle and sophisticated. When you get stuck (and you may be happy indeed with your progress if you do not), please do not give up. Plough on, skim a little, look at the applications of the parts you do not understand, then come back and try again and again. You will get there in the end; and with luck you will have acquired an understanding that Fortran will never give you! Equally, we may sometimes appear to be labouring the most obvious points. This will usually be to help the experienced reader get rid of his bad habits – Algol 68 is full of obvious points that were anything but obvious a few years ago.

A note for high priests. This book is not meant for you. We have tried to describe one (our!) practical way of doing things, rather than all the theoretical ways; we felt that the ultra-precision and detail which some (not all, thank goodness) of you insist on would produce an extraordinarily dull book.

Finally, our acknowledgements. We have cribbed shamelessly from everyone, colleagues, students, visitors, correspondents, conference delegates, friends, relations, Uncle Tom Cobleigh and all, who has aired thoughts about Algol 68 in our presence. Our thanks to them all, but particularly to

– Sue Bond, Peter Hibbard, Richard Housden, Charles Lindsey, Brian Meek and Philip Woodward, who read part or all of sundry versions of our manuscript (but bear little or no responsibility for its current state – except BLM, who perpetrated at least two of the 'jokes'),

– Joan Griffiths, Evelyn Lawes, Peggy Scheppele and (most nobly, since she had first to learn how to use the Unix operating system whereby our manuscripts were prepared and edited) Anne Jennings for their patient typing;

– Julie Brailsford, joint chief guinea pig (with Anne Jennings);

– Larry Hanes and Alan Robertson, who converted and ran our programs on the Cyber 175 at the University of Illinois;

– and last but by no means least Ellis Horwood. In 1972, we naively believed that the world of publishing was a cosy, somewhat old-fashioned world of gentlemen and leather chairs. Five years later, after adventures too tedious to relate, we thought of it more as a cynical world of espionage and deceit. It was a real pleasure to us to meet, through Brian Meek, a family concern which has restored our faith and which has brought our manuscript to the light of day.

All our programs have been tested using the Algol 68R compiler on the ICL 1906A at Nottingham, the 68S compiler on our departmental PDP 11/34, and the CDC Algol 68 compiler on the Cyber 175 at the University of Illinois, and they *all work*! Nevertheless, it is too much to hope that all errors have been detected. Responsibility for any remaining program bugs or textual mishaps rests firmly with us, and we should be glad to hear of them.

DFB, ANW,
July 1978

Introduction

0.1 ALL YOU NEED TO KNOW ABOUT COMPUTING . . .

This book is about a specific way of programming computers. We are going to present at least the basic parts of a total philosophy which we feel offers a better approach to computing than the prevalent current practice. Why better? Really, just because it is a total philosophy. Algol 68 is far from perfect, but it seems to us to represent the first reasonably successful attempt to make everything 'slot together'. Before Algol 68, computing, like Topsy, just 'growed'; one result being that you had to choose between programs that looked nice and programs that worked efficiently. Now you can have both! We hope we can convince you – if you need convincing – about Algol 68; if we cannot, then at least you should be able to return to your old methods – if any! – with a new outlook.

We have to start somewhere. We assume that you have a reasonable grasp of the material usually presented in a computer appreciation course or textbook. You will find it advantageous to have access to a computer with Algol 68 facilities; but we shall not assume any previous programming experience. Indeed, in some respects, previous experience may even be harmful, because the bad habits picked up while learning can be very persistent – and many of today's bad habits were the standard methods of ten years ago.

If everything in this paragraph makes sense to you, you know enough to read this book. If some concepts are unfamiliar, or only hazily remembered, you should do some preliminary reading; see, for example 'Using Computers', by B. L. Meek and S. Fairthorne (Ellis Horwood, 1977). A computer is controlled by a program, which guides the computer in processing information (data processing). The information is transferred into and out of the computer by means of peripherals, such as card readers and tape readers for input, and line-printers for output. Information may also be transput (input or output) via backing store, such as magnetic tapes, discs or drums. For real time or online work, transput may be to a teletype terminal or to a visual display unit. The hardware of a computer system comprises the computer itself together with its storage devices and peripherals. The software comprises the operating system and all the programs, such as assemblers, loaders and compilers, needed to make the computer usable. The computer contains at least one central processing unit (consisting of a control unit and an arithmetic unit), and a certain amount (perhaps 256K words) of core storage or other memory. The memory is divided into words (storage locations, memory cells), each word having an address. Each word contains a certain amount of information, perhaps 24 bits; the information may, for example, be a number (an integer or a floating point number, usually) or part of a number, expressed in binary notation, or some characters, or an address, or an instruction. It is not usually possible to determine which of these sorts of information a given bit pattern represents. To solve a computing problem, you must write a program and appropriate job control, which may give the operating system such information as your identification, the files to be used for transput, and the amount of storage your program will need. Your program will be written in some language, either low level (conforming closely to the machine code of the computer) or high level (corresponding more to some mathematical or logical notation). A high-level language is translated (compiled) automatically into machine code by its compiler; it will almost certainly provide you with facilities for expressions, loops, conditional branches (jumps), subroutines (functions, procedures, routines) and transput, and it should be much easier to test and debug programs than in low-level languages.

Now read on . . .

0.2 HIGH-LEVEL LANGUAGES

The early history of high-level languages is very confused, with every computer installation developing its own ‘autocode’, each having features pirated from earlier versions (on the same or different computers), features specific to a particular computer, features put in at the whim of the local expert, and so on. Out of the confusion, there emerged three high-level languages which gained general acceptance throughout the computing world, and which indeed are still (regrettably, perhaps) prevalent today, namely Fortran, Algol 60 and Cobol.

Fortran (FORmula TRANslator) was originally developed by IBM for their own computers, but it proved sufficiently popular that nearly every computer possesses a compiler for it. (One of the advantages of high-level languages is that, if the language is reasonably machine independent, it may be possible to write compilers for the same language on different machines, thus greatly easing the problems of transferring programs from one machine to another.) Fortran is (supposedly) an easy language to learn, and it is possible on most computers to compile it into very efficient machine code. It provides for very easy evaluation of formulas, fairly easy construction of programs using loops and conditions, and rather rudimentary storage control, transput and subroutines, all in a form which is somewhat similar to ordinary algebra. It is still very widely used by engineers and scientists for whom efficient evaluation of formulas is a paramount consideration.