Ellis Horwood · Publishers
COMPUTERS AND THEIR APPLICATIONS 28

INTERACTIVE FORTRAN 77

a hands-on approach

Ian Chivers and Malcolm Clark

INTERACTIVE FORTRAN 77
A Hands-On Approach

ELLIS HORWOOD SERIES IN COMPUTERS AND THEIR APPLICATIONS

Series Editor: Brian Meek, Director of the Computer Unit, Queen Elizabeth College, University of London

Atherton, R.	Structured Programming with COMAL
Berry, R.E.	Programming Language Translation
Brailsford, D.F. and Walker, A.N.	Introductory ALGOL 68 Programming
Bull, G.M.	The Dartmouth Time Sharing System
Burns, A.	New Information Technology
Burns, A.	The Microchip: Appropriate or Inappropriate Technology
Cope, T.	Computing using BASIC: An Interactive Approach
Dahlstrand, I.	Software Portability and Standards
Davie, F.J.T. and Morrison, R.	Recursive Descent Compiling
Deasington, R.J.	A Practical Guide to Computer Communications and Networking Second Edition
Deasington, R.J.	X.25 Explained
Ennals, R.	Logic Programmers and Logic Programming
Fossum, E.	Computerization of Working Life
Gray, P.M.D.	Logic, Algebra and Databases
Harland, D.M.	Polymorphic Programming Languages
Hill, I.D. and Meek, B.L.	Programming Language Standardisation
Hutchison, D.	Fundamentals of Computer Logic
McKenzie, J. and Lewis, R.	Interactive Computer Graphics in Science Teaching
Matthews, J.	FORTH
Meek, B.L. and Fairthorne, S.	Using Computers
Meek, B.L., Heath, P. and Rushby, N.	Guide to Good Programming Practice, 2nd Edition
Millington, D.	Systems Analysis and Design for Computer Application
Moore, L.	Foundations of Programming with PASCAL
Pemberton, S. and Daniels, M.	PASCAL Implementation
Pesaran, H.M. and Slater, L.J.	Dynamic Regression: Theory and Algorithms
Sharp, J.A.	Data Flow Computing
Smith, I.C.H.	Microcomputers in Education
Spath, H.	Cluster Analysis Algorithms
Spath, H.	Cluster Dissection Algorithms
Stratford-Collins, M.J.	ADA: A Programmer's Conversion Course
Teskey, F.N.	Principles of Text Processing
Thimbleby, H.	Principles of User Interface Design
Turner, S.J.	An Introduction to Compiler Design
Young, S.J.	An Introduction to ADA
Young, S.J.	Real Time Languages

ELLIS HORWOOD BOOKS IN COMPUTING

Atherton, R.	Structured Programming with BBC BASIC
Barrett, T.P. and Colwill, S.	Winning Games on the Commodore 64
Barrett, T.P. and Jones, A.J.	Winning Games on the VIC-20
Christensen, B.	Beginning COMAL
Cole, D.G.J.	Getting Started on the ORIC-1
Cotton, K. et al.	Information Technology and the New Generation
Ennals, R.	Beginning micro-PROLOG
Goodyear, P.	LOGO: A Guide to Learning through Programming
Jones, A.J. and Carpenter, G.	Mastering the Commodore 64
Jones, A.J., Coley, E.A. and Cole, D.G.J.	Mastering the VIC-20
Matthews, T. and Smith, P.	Winning Games on the ZX Spectrum
Moore, L.	Mastering the ZX Spectrum
Narayanan, A.	Beginning LISP
Simon and Matthews, J.	Mastering the Electron

INTERACTIVE FORTRAN 77
A Hands-On Approach

I.D. CHIVERS, B.Sc., C.Ed.
Senior Analyst and Programmer,
Imperial College, University of London

M.W. CLARK, B.A., M.Sc.
Senior Analyst and Programmer,
Imperial College, University of London

ELLIS HORWOOD LIMITED
Publishers · Chichester

Halsted Press: a division of
JOHN WILEY & SONS
New York · Chichester · Brisbane · Toronto

First published in 1984 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 4001, 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
Chivers, I.D.
Interactive Fortran 77.-
(Ellis Horwood series in computers and their applications)
1. FORTRAN (Computer program language)
2. Interactive computer systems
I. Title II. Clark, M.W.
001.64'25 QA76.73.F25

Library of Congress Card No. 84-10890

ISBN 0-85312-775-1 (Ellis Horwood Limited)
ISBN 0-470-20101-0 (Halsted Press)

Printed in Great Britain by R.J. Acford, Chichester.

Table of Contents

Preface

The aim of this book is to introduce the concepts and ideas involved in problem solving with Fortran 77 using an interactive timesharing computer system. The book tries to achieve this using the established practices of *structured* and *modular* programming. Two techniques of problem solving, so-called *top-down* and *bottom-up* are also introduced.

The book has been developed from a one week full-time course on programming, given several times a year at Imperial College to a variety of students, both undergraduate and postgraduate. The course itself is a mixture of

- Lectures
- Tutorials
- Terminal sessions
- Reading

All work on the course is done in small groups, and the students have the option of working in pairs. Initially, students are shy about showing their ignorance, but quickly overcome this and learn a lot by helping one another out and articulating their problems. This is regarded as an essential part of the course.

The student is assumed to complete a minimum number of the problems. Experience on courses over several years has shown the authors that only by completing problems fully does the student get a realistic idea of the process of problem solving using a programming language. It is therefore recommended that all problems attempted are completed. Certain of the problems are used as a basis for further development in the course. This helps to reinforce the ideas of problem solving introduced earlier.

The authors are pleased to provide more details of the course to interested parties.

to Joan
to Glasgow
'Flourish'

1

Introduction to computing

'Don't Panic'

Douglas Adams, 'The Hitch-Hiker's Guide to the Galaxy'

Aims

The aims of this chapter are to introduce the following:–

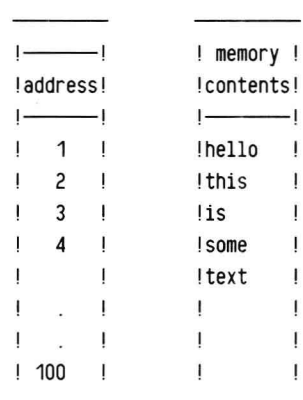
- the components of a computer — the hardware
- the component parts of a complete computer system — the other devices that you need to do useful work with a computer
- the software needed to make the hardware do what you want

A computer

A computer is an electronic device, and can be thought of as a tool, like the lever or the wheel, which can be made to do useful work. At the fundamental level it works with *bits* (binary digits or sequences of zeros and ones). Bits are often put together in larger configurations, e.g. 6, 8, 16, 32, 60, or 64. Hence computers are often referred to as 8-bit, 16-bit, or 60-bit machines. Most computers consist of the following:–

CPU — This is the heart of the computer. CPU stands for *central processor unit*. All of the work that the computer does is organised here.

MEMORY — The computer will also have a memory. Memory on a computer is a solid state device that comprises a collection of bits/bytes/words that can be read or written by the CPU. A byte is generally 8 bits (as in '8-bit bytes'), and a word is most commonly accepted as the minimum number of bits that can be referenced by the CPU. This referencing is called *addressing* . The memory typically contains programs and data. The following diagram illustrates the two ideas of address and contents of the memory at that address.

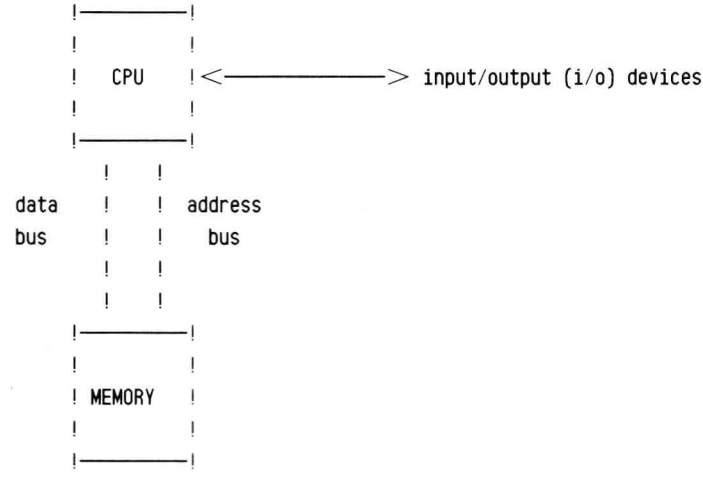


The common word size for a micro-computer is 8 or 16 bits, for a mini 16 or 32 bits and for a mainframe 32, 60 or 64 bits. A computer memory is often called random access memory, or RAM. This simply means that the access time for any part of memory is the same; in order to examine

location (say) 97, it is not necessary to first look through locations 1 to 96. It is possible to go directly to location 97. A slightly better term might have been 'access at random'. The memory itself is highly ordered.

BUS — A *bus* is a set of connections between the CPU and other components. The bus will be used for a variety of purposes. These include control signals to switch parts of the system on or off; address signals which tell the memory which words are wanted next; data lines which are used to transfer data to and from memory, and to and from other parts of the computer system etc. This is typical of many systems, but systems do vary considerably; while the information above may not be true in specific cases, it provides a general model.

A diagram for the constituent parts of a 'typical' computer is given below.



The components of a computer system

So far the computer we have described is not sufficiently versatile. We have to add on other pieces of electronics to make it really useful.

Disks — These are devices for storing collections of *bits*. One advantage of adding these to our computer system is that we can go away, switch the machine off, and come back at a later time and continue with what we were doing. The

memory of a computer is expensive and fast whereas a disk is slower but cheaper. Most computer systems balance speed against cost, and have a small memory in relation to disk capacity.

Tapes — These are slower than disks but cheaper, generally. They vary from ordinary, domestic cassettes used with micros to very large drives found on most mainframe systems. These devices are used for storing large quantities of data.

Others — There are a large number of other input and output devices. These vary considerably from system to system depending on the work being carried out. Most large computer systems have card readers and line-printers whilst other installations may have more sophisticated i/o devices, e.g. plotters for drawing graphs and photo-typesetters for the production of high quality printed material.

The most important i/o device is the terminal. This book has been written assuming that most of your work will be done at a terminal. Terminals tend to come in two main types — those which have a visual display screen, a *vdu* and those which operate rather like a typewriter, and type out onto a roll of paper (tele-typewriter or *tty*). In either case you communicate through the keyboard. This looks rather like an ordinary typewriter keyboard, although some of the keys are different. However, the location of the letters, numbers and common symbols is fairly standard. Don't panic if you have never met a keyboard before. You don't have to know much more than where the keys are. Even professional programmers seldom advance beyond the stage of using two index fingers and a thumb for their typing. You will find that speed of typing is rarely important, it's accuracy that counts.

One thing that people unfamiliar with keyboards often fail to realise is that what you have typed in is not sent to the computer until you press the *carriage return* key. To achieve any sort of communication you must press that key; it will be somewhere on the right hand side of the keyboard, and will be marked 'return', 'c/r', 'send', 'enter', or something similar.

Software

So far we have not mentioned software. Software is the name given to the programs that *run* on the hardware.

Programs are written in *languages*. Languages are commonly divided into two categories; *high-level* and *low-level*. A low level language (e.g. assembler) is closer to the hardware, while a high level language (e.g. Fortran) is closer to the problem statement. There is typically a one to one correspondence between an assembly language statement and the actual hardware instruction. With a high level language there is a one to many correspondence; one high level statement will generate many machine level instructions.

A certain amount of general purpose software will have been written and distributed by the manufacturer. This software will typically include the basic operating system, several *compilers*, an *assembler*, an *editor*, and a *loader* or *link editor*.

> A *compiler* translates high level statements into machine instructions;
> An *assembler* translates low level or assembly language statements into machine instructions;
> An *editor* makes changes to another program.
> A *loader* or *link editor* takes the output from the compiler and completes the process of generating something that can be executed on the hardware.

These programs will vary considerably in size and complexity. Certain programs that make up the operating system will be quite simple and small (like copying utilities), while certain others will be very large and complex (like a compiler).

In this book we concentrate on software or programs that you write for your course, research, or work. As the book progresses you will be introduced to ways of building on what other people have written, and how to take advantage of the vast amount of software already written, tested and documented.

Operating systems

There are generally a variety of operating systems available for a particular computer. The choice of operating system will depend on the kind of work that the computer system has to do. A timesharing operating system is one of the best for general purpose problem solving. These systems allow tens or even hundreds of users to use the system simultaneously. Rapid feedback is

possible, and you can model complex systems, interact with the model, and even change the model, sometimes in a matter of minutes. It is also possible to set up a problem quickly and have it run as a background process, whilst you work on another aspect of the problem.

Problems

1. Distinguish between a memory address and memory contents.

2. What does RAM stand for?

3. What would a WOM (write only memory) do? How would you use it?

4. What does CPU stand for? What does it do?

2

Introduction to problem solving

Aims

The aims here are to:–

- introduce the idea of an algorithm
- introduce two ways of approaching algorithmic problem solving — top-down and bottom-up
- show the difference between the two with a concrete example
- stress the need for pencil and paper study before using a terminal

Algorithms

An algorithm is a sequence of steps that will solve part or all of a problem. One of the most easily understood examples of an algorithm is a recipe. Most people have done some cooking, if only making toast and boiling an egg!

A recipe is a sequence of things to be done. There is an order to be followed. There is no point serving up the vegetables if they have not been prepared or cooked. However certain things can be done in any order — it may not make any difference if you prepare the potatoes before the carrots.

There are two common ways of approaching problem solving, using a computer. They both involve *algorithms* , but are very different from one another. They are called top-down and bottom-up.

In a 'top-down' approach the problem is first specified at a high or general level — prepare a meal. It is then refined until each step in the solution is explicit and in the correct sequence, e.g. peel and slice the onions, then brown in a frying pan before adding the beef. One draw-back to this approach is that it is very difficult to teach to beginners because they rarely have any idea of what 'primitive' tools they have at their disposal. Another drawback is that they often get the sequencing wrong, e.g. 'now place in a moderately hot oven' is frustrating firstly because you may not have lit the oven (sequencing problem), and secondly because you may have no idea how hot 'moderately hot' really is. However as more and more problems are written it becomes one of the most effective methods for programming.

Bottom-up is the reverse to top-down(!). As before you start by defining the problem at a high level (prepare a meal). However, now there is an examination of what tools are available to solve the problem. This method lends itself to teaching since a repertoire of tools can be built up and more and more complicated problems can be tackled. Thinking back to the recipe, there is no point trying to cook a 6 course meal if the only thing that you can do is boil an egg and open a tin of beans. The bottom-up approach has advantages for the beginner. However, there may be a problem when no suitable tool is present. One of the authors learned how to make Bechamel sauce, and was so pleased by his success that every other meal had a course with Bechamel sauce. Try it on your fried eggs one morning. Here was a case of specifying a problem 'prepare a meal', and using an inappropriate but plausible tool 'Bechamel sauce'.