COMPUTING WITH
FORTRAN
A PRACTICAL COURSE

Donald M. Monro

COMPUTING WITH FORTRAN
A PRACTICAL COURSE

Donald M. Monro
Department of Electrical Engineering
Imperial College of Science and Technology

Edward Arnold

First Published 1977
by Edward Arnold (Publishers) Ltd.,
25 Hill Street, London W1X 8LL.

ISBN: 0 7131 2546 2

Printed in Great Britain by The Pitman Press Ltd. Bath.

Preface

At Imperial College developments in the computing services made it possible to develop courses intended to give real experience in problem solving by computer to undergraduate students at a level unattainable by traditional programming courses. Having completed an introduction to computing using BASIC*, I turned to the ultimate objective of fluent FORTRAN and looked unsuccessfully for a suitable structured text before preparing the course which has grown into this book.

The traditional intensive FORTRAN course was defeated by lectures, coding forms, and poor turnaround, all of which divorced the student from his programming. By contrast I prefer to give only an introductory lecture and set the class loose on the facilities with assistance supplied and instructions to submit certain solutions by certain deadlines. The student prepares and runs his own programs and in my mind this is the important difference which does not require a timesharing service to make it work.

Almost every experienced programmer claims to be self-taught, and that is why I intend this book not as a teaching aid, but a learning aid. It is structured to the extent allowed by the nature of FORTRAN and endeavours to stress style and efficiency, while introducing many techniques and methods used in practice. The FORTRAN is essentially FORTRAN IV but some nonstandard features are too good to omit and some compilers are so restricted that the alternatives have to be outlined. I am well aware that a new standard FORTRAN is imminent but it will take some years to apply widely and we cannot wait for that. I take some care to point out common pitfalls and if some are overemphasized it may be because I once stumbled badly there myself.

Chapters 2 to 5 constitute a good grounding in practical application of FORTRAN to data processing and numerical computation. As in BASIC* there is a strong emphasis on numerical methods and this is taken to a more advanced level. This should not defeat the student aimed at science or engineering because I have tried to treat these as exercises in computing, not mathematics with the intention of making the computing interesting, even challenging.

*Monro, D. M., *Interactive Computing with BASIC, A First Course*, Edward Arnold, London (1974)

I am grateful to Professor John Brown and Dr. D. Jones for allowing my approach to be developed on real students, and I am indebted to my colleagues J.M. Howl and P.R. Mason for helping to see the course through its first two years while protecting the students from my worst excesses. I have been fortunate in the help of Mary Mills who patiently typed her way through innumerable drafts, and Linden Rice has shown incredible tolerance in carefully preparing the final version in the face of many changes and delays.

1976

D. M. Monro
Imperial College, London

The Statements of FORTRAN

Items in square brackets are optional

iv = integer variable   ivc = integer variable or constant   sn = statement number
list = list of variable names subject to differing subscripting rules, see text
iunit = unit number, unsigned integer variable or constant

variable = expression	**Chapter 2, 6**
ASSIGN *sn* **TO** *iv*	**Chapter 3**
BACKSPACE *iunit*	**Chapter 7**
BLOCK DATA	**Chapter 5**
CALL *name* [(*arguments*)]	**Chapter 4**
COMMON [/*name*/] *list* [/*name*/*list* . . .]	**Chapter 5**
COMPLEX *list*	**Chapter 6**
CONTINUE	**Chapter 3**
DATA *list/values/*[,*list/values/* . . .]	**Chapter 5**
DIMENSION *list*	**Chapter 5**
DO *sn iv = ivc, ivc* [,*ivc*]	**Chapter 3**
DOUBLE PRECISION *list*	**Chapter 6**
END	**Chapter 2**
END FILE *iunit*	**Chapter 7**
ENTRY *name* [(*dummy arguments*)]	**Chapter 4**
EQUIVALENCE (*list*) [,(*list*) . . .]	**Chapter 5**
EXTERNAL *name* [,*name* . . .]	**Chapter 6**
FORMAT (*specification*)	**Chapter 2, 3, 4, 6, 7**
FUNCTION *name* (*dummy arguments*)	**Chapter 4, 6**
GO TO *sn*	**Chapter 2**
GO TO (*sn* [,*sn* . . .]), *iv*	**Chapter 3**
GO TO *iv* (*sn* [,*sn* . . .])	**Chapter 3**
IF (*arithmetic expression*) *sn, sn, sn*	**Chapter 3**
IF (*logical expression*) *statement*	**Chapter 3, 6**
INTEGER *list*	**Chapter 6**
LOGICAL *list*	**Chapter 6**
NAMELIST/*name*/*list* [/*name*/*list* . . .]	**Chapter 7**
PRINT *sn* [,*list*]	**Chapter 7**
PUNCH *sn* [,*list*]	**Chapter 7**
READ (*iunit, sn*) [*list*]	**Chapter 2, 5, 7**
READ (*iunit, name*)	**Chapter 7**
READ (*iunit*) *list*	**Chapter 7**
REAL *list*	**Chapter 6**
RETURN	**Chapter 4**
REWIND *iunit*	**Chapter 7**
STOP	**Chapter 2**
SUBROUTINE *name* [(*dummy arguments*)]	**Chapter 4**
WRITE (*iunit, sn*) [*list*]	**Chapter 2, 5, 7**
WRITE (*iunit, name*)	**Chapter 7**
WRITE (*iunit*) *list*	**Chapter 7**

FORMAT Specifications

w = field width   n = no. of items   d = no. of decimal places

*n***I***w integer*   *n***F***w.d real without exponent*
*n***E***w.d real with exponent*   *n***D***w.d double precision (with exponent)*
*n***L***w logical*   *n***O***w octal*   *n***Z***w hexadecimal*
*w***H** *literal*   *n***A***w alphanumeric*   *w***X** *spaces*
/ new line   *n* **(** *. . .* **)** *repeated group*   *n***P** *scaling*

Contents

CHAPTER 7 - INPUT AND OUTPUT

1 Introduction

1 About FORTRAN

Those who invent acronyms have had more practice since the phrase FORmula TRANslation was compressed into FORTRAN. It began as a simpler, more restricted language but in its present form, known as FORTRAN IV, it has settled down as the most common general purpose vehicle for data processing and numerical calculation in science and engineering. Many special purpose languages have sprung up ideally suited to a bewildering variety of tasks, but none presents any particular difficulty in learning after FORTRAN. Therefore FORTRAN is the one computer language most worth knowing outside the commercial field (where COBOL prevails).

Experience with FORTRAN has naturally brought an awareness of its shortcomings and no effort is made here to conceal these. One important consideration that reveals itself through trying to learn it and later in helping others is that FORTRAN is not the ideal language for a complete beginner because a large body of complex rules applies to even the simplest program. In this connection a special purpose language to mention here is BASIC because it is worth learning first. BASIC enables beginners to assimilate the elementary principles of programming with a minimum of fuss and is designed to facilitate transition to the greater rigour of such languages as FORTRAN. This course has been made general enough for any student of FORTRAN with a suitable mathematical background, but it is particularly suitable to follow BASIC.

2 About Computers

Man has invented many tools which strengthen his powers, and computers are no exception because of their capability for automating the repetitive calculations which earlier inventions have necessitated. Every computer is a machine endowed with a repetoire of simple instructions which it obeys blindly as a result of human guidance. The job of organizing these instructions into a task for the machine is called programming. The finished list of instructions is called a program and is expressed on paper and to the machine in a programming language, often FORTRAN. The computer has no way of knowing whether the instructions given to it make sense or are what the programmer really intended. It interprets them quite literally and could easily get stuck repeating the same meaningless operation until stopped by human intervention or a timing circuit. The person who is trying to get a program to work correctly is capable of many mistakes but (usually) knows his intentions and can deduce what is going wrong. Much of the effort in computer programming is devoted to finding errors in the program.

The machine normally makes no errors but also exercises no judgement. Beginners are quick to blame the computer for making errors when they cannot find them themselves. Be warned, however, that in your first week you will lose count of your own errors but you may never run out of fingers for counting mistakes attributable to the machine itself.

Provided a computer is instructed properly it can outdistance in seconds or minutes a human lifetime of hand calculation. This is why computers have had a profound effect on a bewildered society. The effects are not always beneficial, particularly if a decision to "computerize" is taken in ignorance of the large overheads and highly specialized skills involved. But computers can add a million numbers a second and most can multiply nearly as rapidly with impressive precision. A computer can store thousands or tens of thousands of results in its memory and recall any one of them in a microsecond. It can be programmed to examine its results and make a decision and so can be given a superficial appearance of intelligence - but this intelligence originates with the human programmer and the computer's mistakes nearly always have the same origin.

A computer system is much more than a machine which does calculations. To be useful it must be surrounded by devices which feed it information and it must be given programs to guide it through its tasks. The person learning FORTRAN may communicate through a terminal with a keyboard for him to transmit information to the computer and a printer for its responses. Perhaps less fortunately he may have to use a "batch service" to which punched cards are submitted and from which the results are returned later.

A computer could have connected to it readers and punches for cards and paper tape, magnetic tape transports, lineprinters, and magnetic disk storage. All these devices provide for input (to the computer) and output (from the computer) of information. Each device has a "driver" program to control it, and there will be a supervisor for the drivers (and probably a program to monitor the supervisor). All these devices and programs make up a computer system before FORTRAN

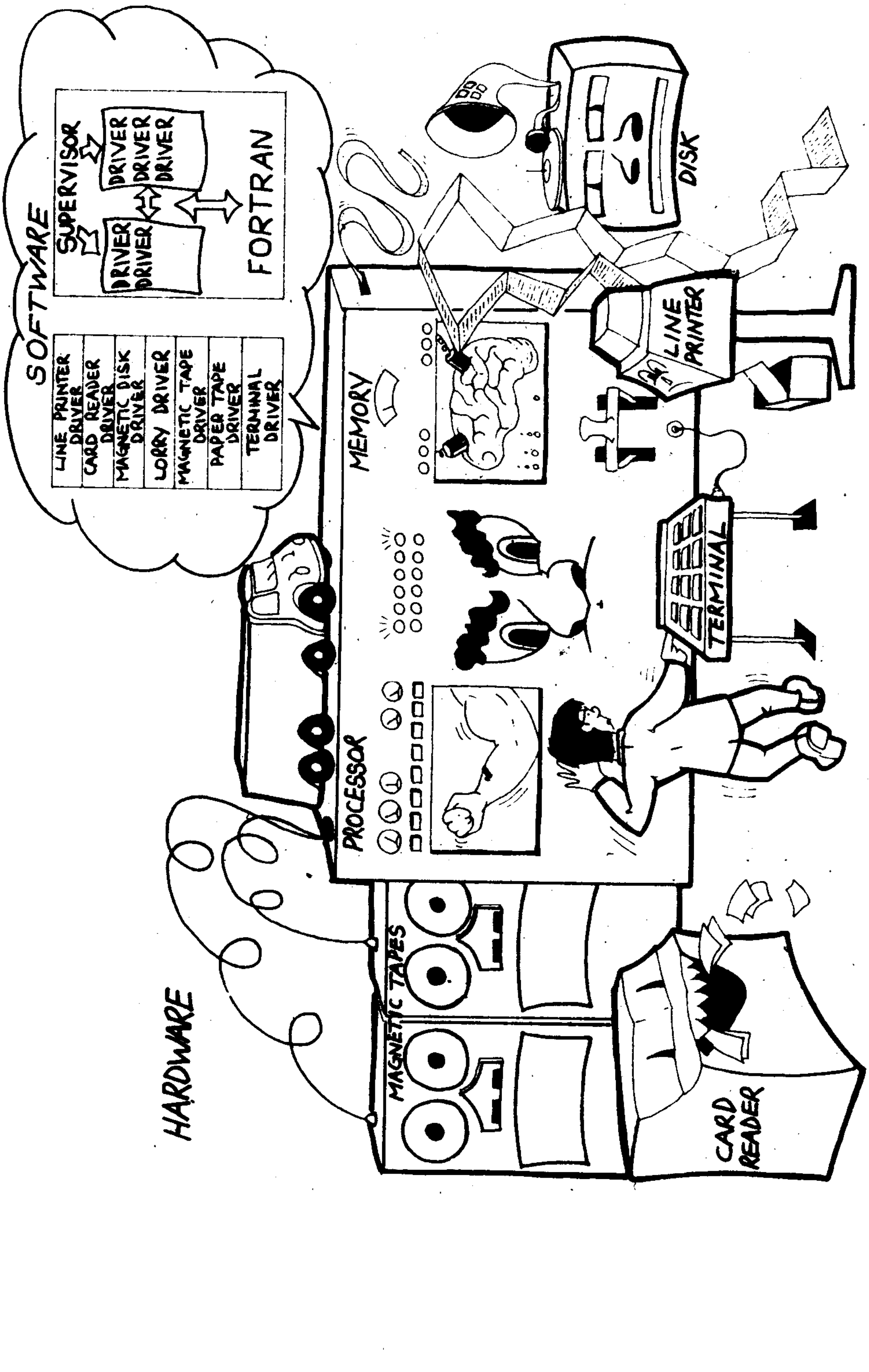

Fig.1.1. A bewildering array of apparatus inhabits the Computer Room.

is taken into consideration and certainly before the "user" arrives to try his program provided of course that 'they' will let him get near it (Fig. 1).

The computer itself will not understand FORTRAN - the language it takes instructions from is a rather nasty series of numbers. Therefore a translation program or "compiler" is needed which takes a FORTRAN program and converts it to machine language. Because of the many facilities of FORTRAN and the need to check the grammar of a FORTRAN program, the compiler is quite a large program. Thus it takes many programs to run a FORTRAN program and the computer system that supports a FORTRAN programmer is an imposing collection of machinery ('hardware') and programs ('software'). The beginner is protected to an extent from any need for detailed knowledge of all this, but FORTRAN is a language that enables the expert to expand into many of the facilities of the system.

3 Interactive Computing and Time Sharing

Early computer systems were organized to deal with one program at a time, and programs were normally presented to the system in groups or 'batches' which the machine processed one after the other. The programmer submitted his program to a computing service which ran it for him and returned the result some time later. FORTRAN like any other language can be run in this way, and the majority of computing is still done in batches. The disadvantage of batch processing for small programs and for learning is that the 'turnaround' time is unlikely to be less than a few hours and is more likely to be measured in days.

An interactive computer system puts the programmer into direct communication with the computer, usually through a typewriter terminal Therefore the turnaround time for developing programs and finding and correcting errors is reduced to seconds. The program itself can be written so that the programmer gives it information while the computer is executing it and so he can control the steps of the calculation as it progresses. When FORTRAN is run in an interactive system, programs can be developed rapidly and tested and corrected from a terminal. The learning process is both shortened and made more thorough because the rapid response of the computer and the straightforward nature of the language work in the student's favour and encourage experimentation.

An interactive computer system can be in one of two modes of operation as seen by the programmer. These modes are 'program definition' and 'program execution' and are distinguished by

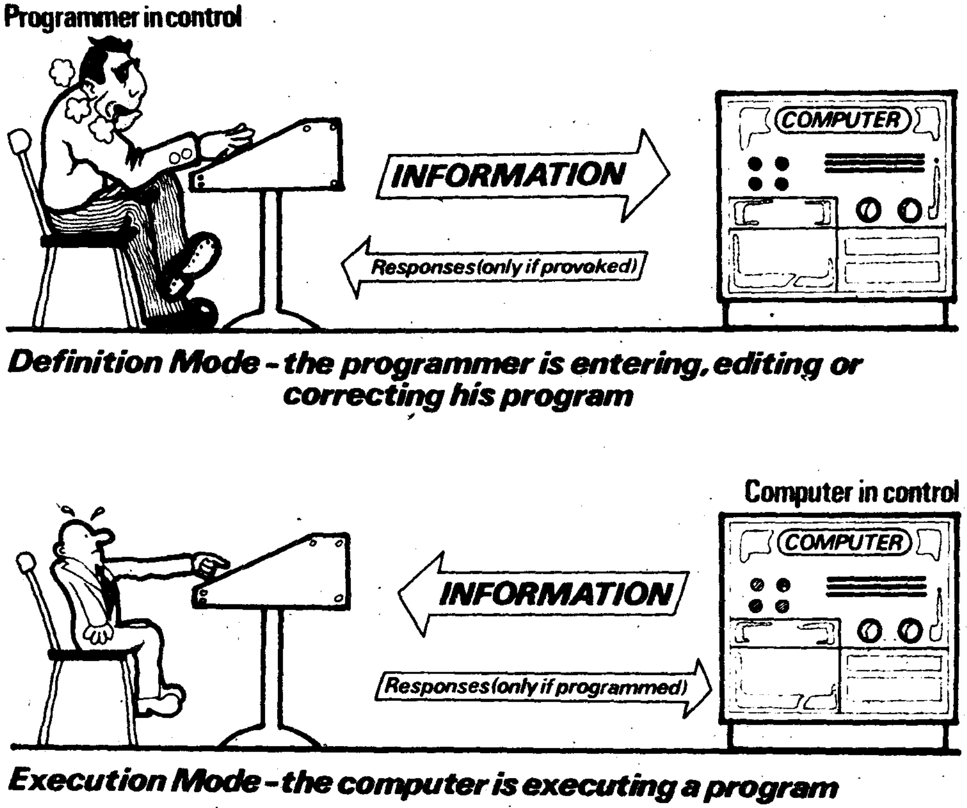

Fig.1.2. Timesharing divides into two modes of interaction.

whether the programmer or the computer is in control of events as in Fig. 2. In the program definition mode the programmer will be creating, editing and correcting his program and is himself in control. The main flow of information is from the terminal to the computer and any response by the computer is a result of the programmer's activities. He can enter commands to the system, and the effect of some of these commands is to transfer control to the computer. If this is done, the system will change to execution mode and the user will be required to respond only if the program has made specific provision for input from the terminal. The main flow of information will be from the computer to the terminal, and the programmer normally will regain control when the program is finished, although he can stop execution manually if necessary.

Time sharing is a means of making the resources of one computer system serve the needs of many users at the same time. The computer does not do several things at once, but it can be made to jump from one task to another so rapidly that the individual user is not aware of any long delays. Therefore interactive computing can be carried out at many terminals 'simultaneously'. Large time sharing systems can service a hundred or more terminals all using a variety of languages to perform different operations, and also can do batch and other work at the same time.

4 Batch Processing

The most likely form of computer service is the batch processing arrangement which, although not ideal for learning, is efficient for the production work which accounts for most computer usage. In this kind of bureau, programs are submitted at a central site and are fed to the computer in batches. Some time later, when the machine has completed the batch and moved on to another, the printed results are returned together with the program which will most often have to be corrected and submitted again. The "turnaround" in a batch service is at best a few hours and often overnight even for small jobs and therefore the progress of a person learning FORTRAN can be badly hampered. A service like this involves human intermediaries who organize the input, sort the output and deal with hundreds of moaning users. The user himself is likely to develop a somewhat jaded view of the reception department. It should be remembered that delays in processing are unlikely to be the fault of the unfortunates who staff the reception area; many computers seem to have an uncanny ability to develop sick headaches at the worst possible moment.

Some enlightened computer centres mitigate the delays of batch processing by granting the user himself access to enough equipment to run his own job, and this does tend to create satisfied customers.

Once the initial shock at the idea of allowing users to not only see but *touch* equipment has passed, it is usually found that a well organized "hands-on shop" in which people can read in their own cards and tear off their own output is a success, if untidy at times.

Almost all batch work is done from punched cards, and the deck of cards that makes up a "job" must contain not only all of the FORTRAN program, but also all of the necessary directions to the computer system to make it run,[1] and any data that the program is intended to process. The directions, called "control cards" or "job control language" vary widely between different computers as does the manner of organization of the deck of cards. Typically control cards will be needed to initiate translation or "compilation" of the original source FORTRAN into machine language, to load this "compiled" program, or "object code" into the computer, search the system libraries for missing bits of program, and set it running, or "executing". A complicated job may involve many more operations.

5 How to Use This Book

The course is intended to be followed from the beginning in order, doing as many problems as possible. It is necessary to have a means of running FORTRAN programs on a computer, ideally by access to an interactive system; if only batch processing is available it will take longer. If possible a source of expert advice should be available, about FORTRAN because people who have made all the mistakes already spot them more quickly (this is called "experience"), and about the computer system which is likely to give more trouble than the FORTRAN at first. The supplementary problems at the end of most chapters are more demanding and should be regarded as optional.

Each section should be read through before problems associated with it are tried, and even the most tentative outline solution to a problem will save time spent on the computer. It is tragic to watch year after year the amount of time wasted in reading the material for the first time and trying to think out the solutions at the keyboard; the same people often claim to have had insufficient access.

A good introduction to practical computing is formed by Chapters 2 to 5, each of them requiring about ten hours of real work. If it takes less, so much the better, but if it requires more then either preparation and organization are inadequate or the level of the course is inappropriate to the particular student's background and interest.

2 Calculations in Fortran

1 Introduction

A computing machine is directed by a series of instructions telling it exactly what to do at each stage of a calculation; a set of these instructions is called a program. Programming languages are used to express instructions in a way which is independent of the minute details of operation of the computer. FORTRAN IV is called a "high level" language because it expresses calculations in terms familiar to humans rather than machines. A FORTRAN program uses common English terms and mathematical operations. However, because the communication is with a computer, the instructions given must be precise and no ambiguities can be allowed. Therefore the grammar of FORTRAN, like any other computing language, is constrained by a precise set of rules which control what grammatical constructions or "syntax" the machine will "understand", i.e. accept as valid instructions. These rules may make FORTRAN look complicated at first, but they are there for good reasons, and experience provides an easy fluency with the language because the rules make sense. This is one of the reasons why FORTRAN is the universal language of scientific calculation and has endured as such for many years. In this chapter enough basic grammar and construction is introduced to allow simple calculations to be undertaken, although some of the material will have to be elaborated on later.

A very simple example of a complete self-contained program serves to introduce the language and point out some of the features of construction. The following program calculates and prints the value of π using the fact that $\tan(\pi/4) = 1.0$:

```
      PIE=4.Ø*ATAN(1.Ø)
      WRITE(6,2Ø)PIE
   2Ø FORMAT(1X,F1Ø.5)
      STOP
      END
```

Undoubtedly there will be things in the program that look familiar, and others that are partially self-explanatory. FORTRAN uses words of English and some recognizable mathematical notation, but it also has very strict rules of punctuation.

The program structure in FORTRAN is straightforward enough. Each line of the program is called a "statement" of FORTRAN, and some lines include statement numbers, as does statement 20 in this example. When a FORTRAN program is executed by a computer, the order of the statements dictates the order in which instructions given in the program are obeyed by the computer. So the given example is executed line by line just as one would read it; here the statement number itself does not affect the order.

This program contains several kinds of statements and other features which must all be understood before any program can be attempted.

The arithmetic statement

```
PIE=4.Ø*ATAN(1.Ø)
```

may be recognised as a replacement or assignment statement for the variable PIE and * represents the multiplication operation. But why are the decimal points given explicitly in the constants 4.0 and 1.0? If they were left out the program would fail - some computers would reject it outright while others would produce the answer 0. Also important to a certain extent is the spelling of the variable PIE; were it called LIE instead a different result (3) would be produced by this statement.

The output statement

```
WRITE(6,2Ø)PIE
```

and its associated FORMAT

```
2Ø FORMAT(1X,F1Ø.5)
```

are involved with the printing of the result. But what does it all mean? In this particular example the value of PIE is written onto unit number 6 in a format of one space followed by the number right justified in the next 10 spaces with five places of decimals shown. This is not as complicated as it sounds and will be fully explained.