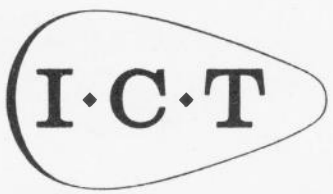
INTERNATIONAL COMPUTERS AND TABULATORS LIMITED

A PRIMER OF FORTRAN PROGRAMMING FOR USE ON ATLAS AND ORION COMPUTERS



LONDON COMPUTER CENTRE,
68 NEWMAN STREET,
LONDON, W.1.

LIST CS 390
OCTOBER 1963

CONTENTS

PREFACE

Compilers have been developed to enable the I.C.T. Atlas and Orion computers to accept programs written in the automatic programming language FORTRAN. This primer is intended to serve as an introduction to the language and should be of use both to the newcomer to programming and to the programmer wishing to obtain an understanding of FORTRAN in particular. The text covers in a progressive manner most aspects of FORTRAN and is illustrated with examples and exercises for which solutions are suggested.

The dialects of FORTRAN described are derived from FORTRAN II, and are very largely compatible with it and with other dialects currently in use. It is anticipated, for instance, that the majority of FORTRAN II routines will require no alteration at all: details of such differences as do exist are to be found in the following document.

CS318B Making a FORTRAN II program suitable for use with the Atlas FORTRAN compiler.

Certain advanced aspects of FORTRAN programming are not covered in this primer and the reader should consult the reference manuals to be provided for full details and formal definitions of the language; these manuals will contain also information concerning the systems of operating FORTRAN on each computer.

ACKNOWLEDGEMENT

This Atlas FORTRAN compiler was written by a team at A.E.R.E., Harwell, Berks. under the leadership of Dr. I.C. Pyle (U.K.A.E.A.,) and this Orion FORTRAN compiler by a team at the Rutherford Laboratory, Chilton, Berks. under the leadership of Dr. R. Taylor (N.I,R.N.S.),

I.C.T. Limited wish to acknowledge the assistance of both establishments in the preparation of this document.

A PRIMER OF FORTRAN PROGRAMMING

FOR USE ON ATLAS AND ORION COMPUTERS

1. INTRODUCTION

FORTRAN (a name derived from FORmula TRANslation) is an automatic programming language for digital computers. It is a complete language in that, as well as providing for the evaluation of formulae and computation of a mathematical nature generally, it includes also facilities of great generality for the input and output of data using many types of peripheral equipment. Dialects of FORTRAN, each embodying the same basic principles although differing in details, have been developed for many computers: it is the purpose of this primer to introduce those available on the Ferranti Atlas and Orion computers. These dialects incorporate a number of features which are either new or included on only a few of the modern large computers.

The programming of solutions to problems in the field of science and engineering is greatly simplified by the use of FORTRAN because it is not necessary to be familiar with details of the internal working of any particular computer. The aim of this primer is to serve as an introduction to programming as well as an introduction to FORTRAN - to this end Section 2 deals with a number of programming concepts in general terms, while in the remaining sections the language is progressively introduced to a sufficiently high level to enable the casual programmer to write successfully fairly complex programs. Numerous examples are included in the text, and exercises and suggested solutions provided; the primer should therefore be suitable for private study or for use on programming courses. The reader should understand that details of finer points of the language and formal definitions, together with information concerning the system of operating FORTRAN on each computer, must be sought from the appropriate reference manual.

The greater power of Atlas, compared with Orion, and the differences between the hardware and the operating systems, lead to a few differences in the implementation of FORTRAN. In general, Orion FORTRAN is a subset of Atlas FORTRAN, in that certain features of the language may not be available on the smaller computer. Furthermore, some features which are to be provided in due course are not included in the first versions of the compilers. Wherever possible, differences have been explicitly stated in the text, or mentioned in a note at the end of a Section.

2. PROGRAMMING CONCEPTS AND TOOLS

2.1 The organisation of a computer

A digital electronic computer is a machine, or a linked set of machines, capable of handling information which can be represented or coded in a numerical form. It can perform at high speed a selection of elementary arithmetic and logical operations on this information in a desired sequence under automatic control.

The computing system may be considered as made up of units each with a different function to perform. An internal central *store* holds the information received into the machine, as groups of binary digits, and holds also other information generated during computation. The actual computing operations are performed in the *arithmetic unit*; this is capable of operations such as addition, subtraction, multiplication and division and of certain logical operations such as comparison and shifting of groups of digits. A *control unit* serves to specify which operations are required, to fetch items from store into the arithmetic unit and to store back the results of the operations.

In order to get information into and out of the computer a variety of *peripheral devices* is available. Information is represented outside the machine on some external medium. For example, patterns of punched holes in cards or paper tape can represent alphabetic and numeric information, and a card- or paper-tape reader is used to read the patterns of holes and to convert the signals received into the internal form required by the computer. Similarly for the output of useful information, devices are used which receive signals from the computer and use these to produce either punched cards or paper tape, or to operate an electric typewriter or a high speed printer.

Information can also be recorded on and read from reels of *magnetic tape* mounted on tape handling units connected to the computer. Because magnetic tapes have a high capacity for information and are comparatively inexpensive, they may be used as an extension of the internal store of the machine; they have the advantage that individual reels of tape may be removed from the handling units and stored away. The information on them is available for subsequent use and may be read back into the computer rapidly when required.

Automatic control of all computer operations is possible because the steps required to perform a particular computation can be worked out in advance and a set of corresponding instructions to the computer can be held in the internal store. This set of instructions is referred to as a *program*. The control unit accepts and calls for instructions from store in sequence, interpreting each one and initiating the appropriate operations. Certain machine instructions are available to call for instructions out of the normal sequence, possibly as a result of some logical comparison of quantities in store, and a

versatile program is thus possible. The complete set of instructions which a particular computer can obey is referred to as its *basic order code* or *instruction repertory*.

2.2 Characteristics of computing tasks

The operating speed of a computer is such that many thousands of instructions can be obeyed every second. Clearly a program to run for any length of time must be of a repetitive nature, so that the number of instructions required in store, as distinct from the number obeyed, is not excessive; tasks suitable for automatic computation have therefore certain characteristics. These are perhaps best illustrated by a few examples.

(a) *Matrix manipulation.* Matrix operations involve a large amount of computation, but this consists of many repetitions of identical calculations on rows and columns in a systematic manner. Provided only that the actual elements of the matrices are themselves stored systematically, a program can be arranged which consists of loops of instructions obeyed repeatedly, referring to the elements in turn in an orderly manner.

(b) *Engineering design.* If the performance of some item of equipment can be represented by a set of equations involving a number of variables, the solution of these for even one case may take some time. To use the computer will save effort, but of greater importance is the fact that the calculation can be repeated automatically for various sets of values of certain of the variables, to permit proper evaluation of the design.

(c) *Analysis of test results.* Experimental work frequently produces large quantities of test data, which requires processing of a tedious and complex kind. Using a computer may be the only way to take full advantage of all the information produced by the test, and to perform adequate statistical analysis.

Numerous other examples could be taken from commercial, mathematical, scientific and engineering fields: construction of tables, payroll, stock control, analysis of structures, solution of sets of differential equations and so on. These various problems have characteristics coming under one or more of the following headings. They may involve:

(i) repetitive and iterative calculation,
(ii) a volume of calculation large in relation to the volume of input data,
(iii) a large volume of input data previously automatically recorded,
(iv) large "files" of data to be maintained and updated for periodic analysis.

Many of the so-called commercial applications have characteristics covered by the last heading.

Because input data must be in a form acceptable to existing input devices, certain problems would not normally be programmed for automatic computation. For example, if several thousand test results were recorded in written form, and the amount of processing required was small, it would not be justifiable to undertake the work of punching the data on to cards or tape in order to occupy a computer for a few seconds. When it is desirable to use a computer for otherwise tedious and time consuming elementary processing some method must normally be found to record the data automatically when it is first obtained.

2.3 Automatic programming languages

In order to write a program to solve a particular problem, a procedure must be devised which can be written in terms of the operations for which instructions exist in the basic order code. Instructions are held in the internal store of the machine as groups of binary digits just as are all other quantities; different combinations of digits denote different operations, and part of the function of the control unit is to decode these combinations and initiate the corresponding operations. Because of its specialised form the order code is not quickly learned and its effective use requires a detailed knowledge of the internal workings of the computer; furthermore, to carry out quite simple calculations the machine may have to obey a whole sequence of instructions, so that writing a program in basic order code is a lengthy task. Generally speaking, such a program will be written by a specialist programmer.

To allow a non-specialist to write programs a number of *automatic programming languages*, such as FORTRAN, have been developed. A problem-solving procedure is written as a series of statements and declarations having a general resemblance to algebraic formulae and English sentences, and a special program used to translate these statements into basic order code. In effect the computer is transformed into a machine with which communication is made in a simpler language than basic order code. The program written in the original language is referred to as a *source program*; the program which performs the translation is the *compiler* or *translator;* the set of machine instructions resulting from the translation is referred to as the *object program.* In the FORTRAN system the translation process is carried out essentially once only: when it is desired to run the program it is the object program which is fed into the computer and which actually reads data and executes the computation.

Among the advantages of using an automatic programming language may be mentioned the following. Firstly, the individual statements in the source program represent quite powerful sections of computation - they may be translated into dozens of machine instructions. Secondly, quantities involved are referred to by convenient names. Thirdly, the forms of statement allowed are easily learned. There is consequently a considerable saving in the time taken to write a program; as a whole the program is a fairly readable document, and alterations in the program are easily and conveniently made. Lastly, the user need know very little of the method of working of the computer, except to have some idea of the merits of the various forms of input and output, and so need not be a computer specialist.

It must be realised that solving a problem using a computer is not simply a matter of writing a program. The problem must be defined and the objectives of the program decided upon; the mathematical methods to solve the problem must be chosen and expressed in terms of the simple operations of which the computer is capable. This is the field of numerical analysis. Furthermore, the computer obeys only the instructions with which it is supplied, and is not capable of exercising judgment unless given explicit instructions to make certain comparisons and decisions. Consequently the procedure devised for solving a problem must be sufficiently detailed to cope with all forseeable circumstances. Finally, it is worth remarking that even when a program is finally written, it is unlikely to contain no mistakes and to work correctly when first tried. The process of translation may show up errors, and the translated program must be thoroughly tested before it is used for regular production work.

2.4 Block diagrams

Regardless of the language used to express a problem-solving procedure, the steps of the procedure must be clearly thought out before the program is written. The next few sections indicate ways in which procedures may be devised which are suitable for automatic computation. No reference is made at this stage to FORTRAN.

An aid to visualising a problem of any complexity is the *block diagram*. This is a diagram made up of a set of boxes of various shapes, each containing a description of some operation, linked by arrows showing the flow of control between operations.

A rectangular box is used to indicate a processing operation and an oval box an input or output operation. A series of such boxes linked one following another indicates a straightforward sequence of operations.

Example 1

A section of program is to read in six numbers a, b, c, d, p, q and evaluate the quantities

$$r = \frac{pd - qb}{ad - bc} \quad \text{and} \quad s = \frac{qa - pc}{ad - bc}$$

A block diagram outlining the necessary steps is shown in Fig. 1.

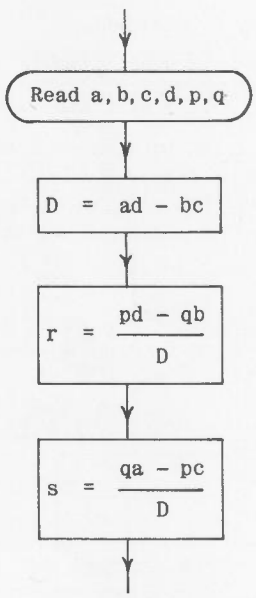


Fig. 1

Note that the expression (ad - bc) is calculated once only.

2.5 Programming Techniques

2.5.1 Decisions. At certain points in all programs it will be necessary to follow different paths of computation according to the results of comparisons or decisions. A diamond-shaped box is used to indicate a decision.

Example 2

In example 1 the calculation will break down if ad - bc = 0. Suppose that the program is intended to check this value, and to branch to a part of the program termed Error if it is zero. Then the block diagram can be enlarged as in Fig. 2.

The paths leaving the decision box are labelled to indicate the nature of the decision. A circular box is used to indicate a connection to another part of the diagram.

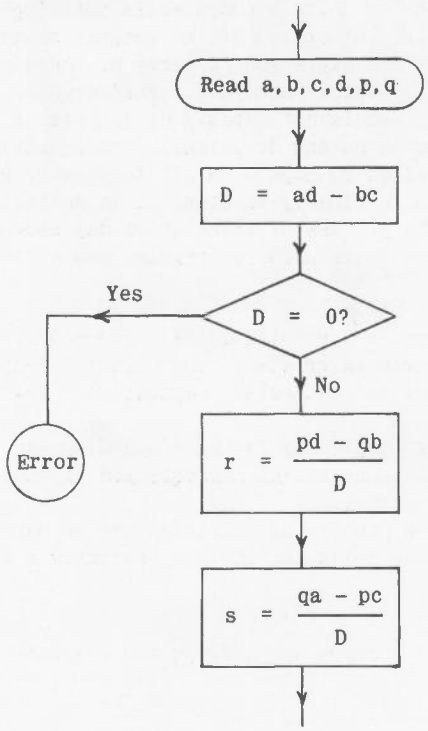


Fig. 2

Example 3

At a point in a program it is required to evaluate

$$\lambda = \left[\frac{2\sin\theta - \tan\theta}{1 + \sin\theta\tan\theta}\right]^2 - 1$$

for some previously calculated value of θ lying between $-\pi/2$ and $+\pi/2$. The programmer will be satisfied with the accuracy obtained by assuming $\sin\theta = \tan\theta = \theta$ if $\theta \leqslant .05$ radians, and small values of θ are expected to occur sufficiently often to make the time saved by using this approximation significant.

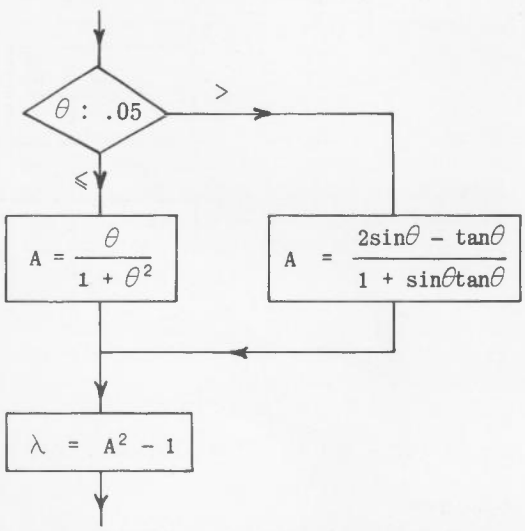


Fig. 3

The colon in θ : .05 is used to indicate a comparison, and again the result of the decision is shown by labelling the exit paths.

Example 4

A program is to follow one of three different paths according to the value of $(b^2 - 4ac)$, which may be negative, zero or positive.

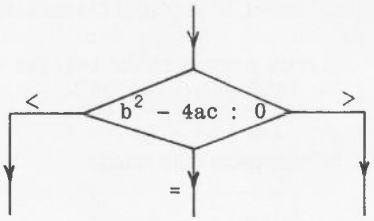


Fig. 4

2.5.2 Loops. A sequence of instructions obeyed repeatedly is termed a *loop*.

Example 5

A program is to read a set of three numbers b, c, A ($A \neq 0$) and to calculate and print the values of $a = \sqrt{(b^2 + c^2 - 2bc\ \cos A)}$ and $\Delta = \frac{1}{2}\ bc\ \sin A$; it is to repeat these operations for further sets of three numbers.

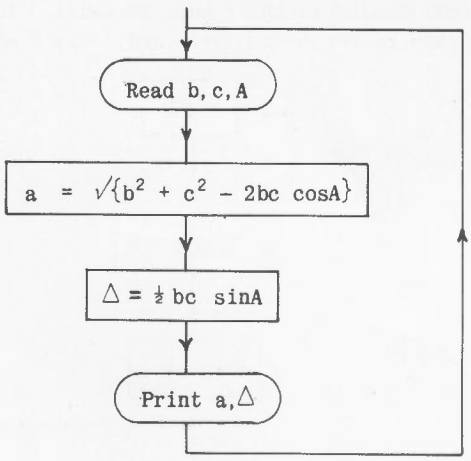


Fig. 5

Clearly this program will never leave the loop, and will come to a stop when no further sets of three numbers are available for reading at the input device. This form of program is not recommended, since a program in general should always follow through particular paths from the start to some recognised and definite end.

Example 6

One method of leaving this particular loop is to read a set of three numbers which are all zero, after the required sets have been dealt with. The program can test for A = 0 and terminate when this value appears.

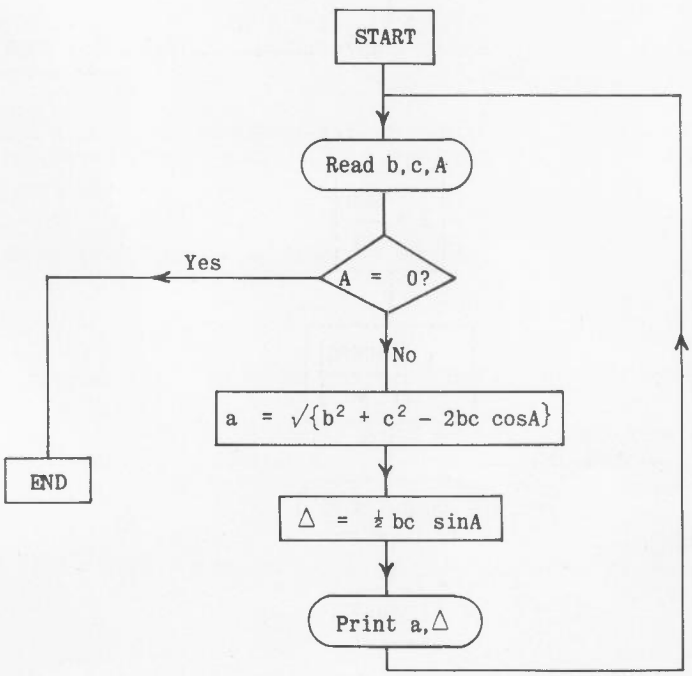


Fig. 6

In this diagram square boxes have been used to indicate starting and stopping points of the program.

2.5.3 Counters. Another method of leaving a loop is by testing and incrementing a *counter* each time round the loop.

Example 7

80 numbers $x_1, x_2, \ldots, x_{80}$ are to be read and the means

$$\bar{x} = \frac{1}{80}(x_1 + x_2 + .. + x_{80})$$

and

$$X = \frac{1}{\sqrt{80}}(x_1^2 + x_2^2 + .. + x_{80}^2)^{\frac{1}{2}}$$

calculated. A possible block diagram is shown in Fig.7.

Two quantities, called here SUM and SUMSQ, are accumulated; these are zero at the start and as each number is read in, its contribution to these sums is added. The program sets up a counter i which runs from 1 to 80; when i reaches 80 the loop is left and $\bar{x}$ and X are calculated.

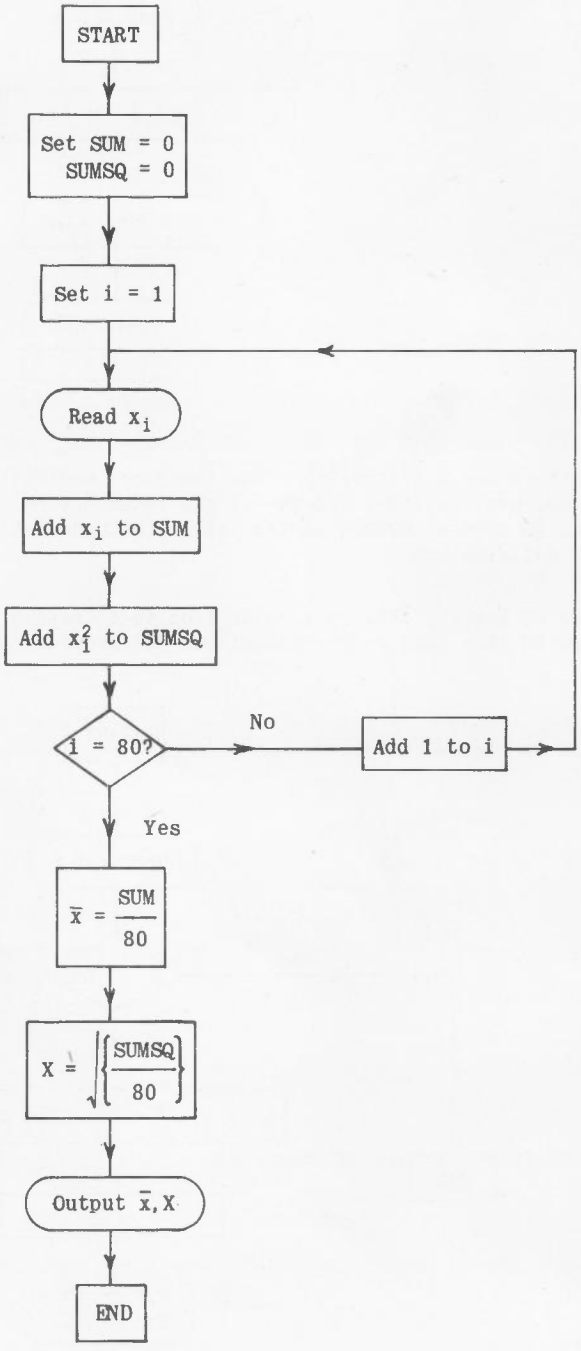


Fig. 7

Phrases such as "Add 1 to i" are clumsy to write. It is customary to write "i = i + 1" having the meaning "set a new value of i equal to the old value of i plus 1". Similarly, it is usual to write "SUM = SUM + x_i" and "SUMSQ = SUMSQ + x_i^2".

Example 8

It is required to print a table of values of n and the corresponding values of $m = 100\ n^k$, for the integral values of n running from 1 to N, N being read in by the program. k is to be determined from a value M (< 100) also read in, so that m = M for n = 2.

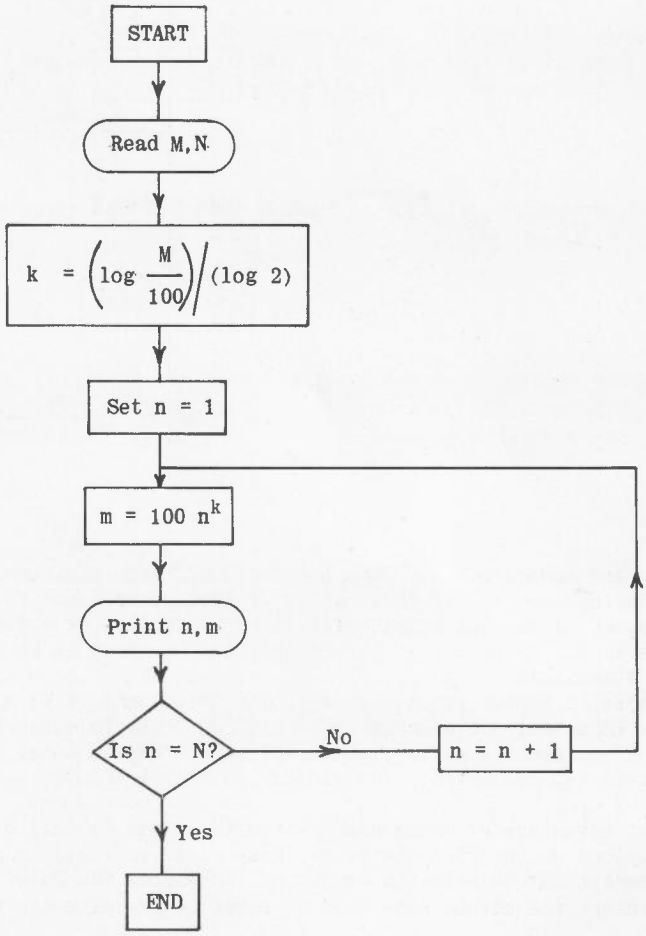


Fig. 8

In this example there is no need for a separate counter, since the variable n itself fulfils this function. The number of repetitions of the loop is not fixed, but is specified in the input data to the program.

2.5.4 Iteration. A loop may be an iterative loop, in which a calculation is repeated until sufficient accuracy has been achieved according to some criterion; the loop is then left and the program proceeds to the next section.

Example 9

Part of a program is to calculate e from the series

$$e = 1 + \frac{1}{1!} + \frac{1}{2!} + \frac{1}{3!} + \ldots.$$

by direct summation of successive terms up to and including the first term to have magnitude $\leqslant 10^{-8}$, all subsequent terms being dropped.

The sum of the first two terms is obviously 2 and each subsequent term is derived from the previous one simply by dividing by a counter n.

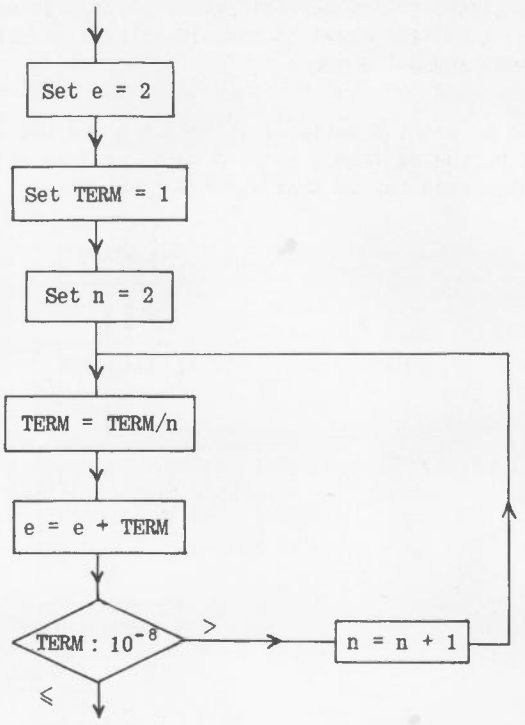


Fig. 9

2.5.5 Arrays and subscripts. The power of automatic programming languages is increased by facilities provided for using *subscripts*; these allow the use of one name to represent a set of quantities, an individual member of the set being referred to by the use of a subscript or subscripts. The entire set of quantities is called an *array*; each member of the array is an *element*. Arrays may have 1, 2, 3 or perhaps more dimensions.

For example, a vector $(x_1, x_2, \ldots, x_{80})$ may be thought of as a one-dimensional array called x, having 80 elements. To select one element x_i requires a single subscript i. A rectangular matrix B having m rows and n columns is a two-dimensional array, with a total of $m \times n$ elements. To select an element requires two subscripts; the element in the i th row and j th column may be referred to as $B_{i,j}$.

The great advantage of using subscripted variables is that programmed loops may be set up to perform operations on the elements of an array in a systematic manner; at each repetition of a loop different elements may be selected simply by increasing the value of a subscript or subscripts in a systematic manner, the actual name used to refer to the elements remaining unchanged.

Example 10

The scalar product P of two vectors A and B, each with N elements, is required. In mathematical notation, $P = \sum_{i=1}^{N} A_i B_i$.

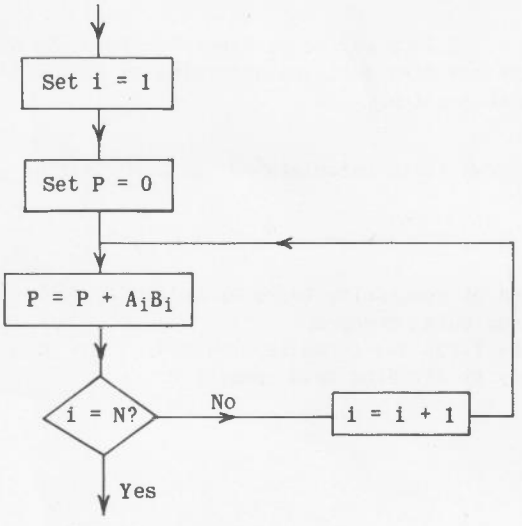


Fig. 10

Example 11

Operations are to be performed on elements of a square array A, size n by n, in which all elements in the upper triangular half are zero, that is $A_{i,j} = 0$ when $j > i$. Obviously the elements of A could be stored as a two-dimensional array and referred to straightforwardly by two subscripts, but this is a waste of storage space if the program takes advantage of the known zeros and avoids referring to them. To save space the non-zero elements only can be stored as a one-dimensional array in the order

$$A_{1,1},\ A_{2,1},\ A_{2,2},\ A_{3,1},\ \ldots,\ A_{n,n}\ .$$

This has $\frac{1}{2}n(n+1)$ elements and an element $A_{i,j} (i \leqslant j)$ occupies the kth position where $k = \frac{1}{2}(i-1) + j$. Thus element $A_{i,j}$ may be referred to simply as A_k; this is an example of the use of a subscript which is supplied as an arithmetic expression rather than as a single integer.

2.5.6 Nested loops. Loops within loops are referred to as nested loops, and are of frequent use.

Example 12

The moment of inertia of a certain angle section about a particular axis is given by the approximate formula

$$I = \frac{4}{3}d^3t - 2d^2t^2 + \frac{4}{3}dt^3 \quad \text{inches}^4$$

where d,t are dimensions in inches.

It is required to vary t from .05 to .15 in steps of .01 and to vary d from 1.00 to 4.00 in steps of .05, calculating I for all possible combinations of t and d. The answers are to be stored as an array having 61 rows corresponding to the values of d, and 11 columns corresponding to the values of t.

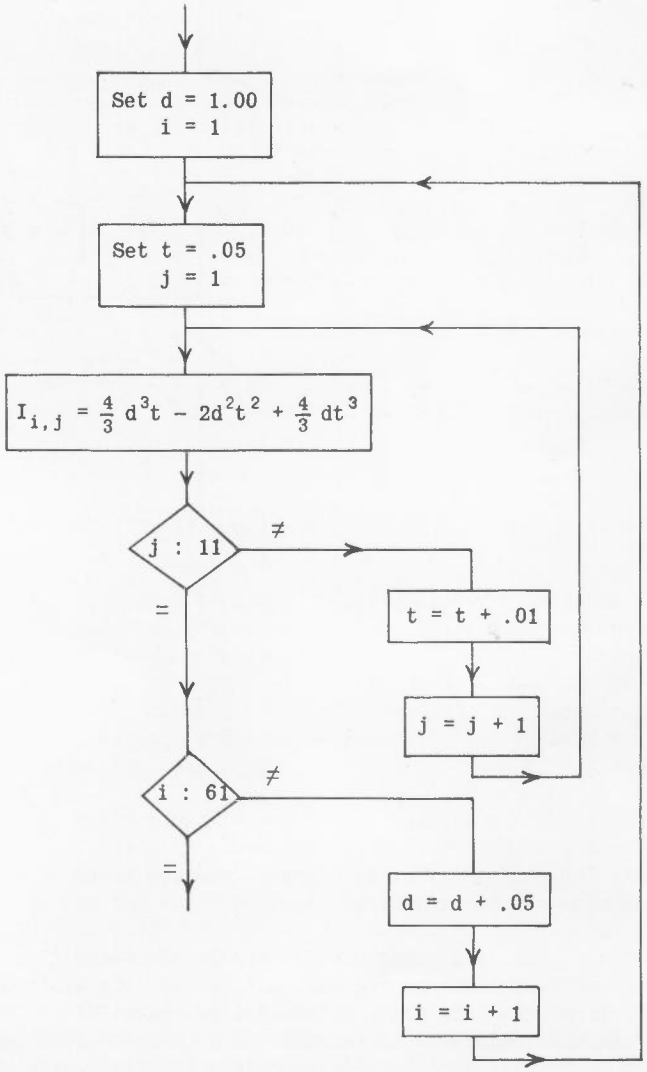


Fig. 11

From Figure 11 it can be seen that the initial values d = 1.00 and i = 1 are first set up, and held unchanged while the inner loop is repeated 11 times with t taking the required values .05, .06, ..., .15 successively. When the inner loop is left for the first time, as determined by the counter j , the elements of the first row of the array have been calculated and stored. The outer loop is then traversed to obtain the next value of d , and to increase the row counter i by 1 ; starting values of t and j for the inner loop are reset and the inner loop traversed again 11 times. This process is repeated until the outer loop has been traversed the required 61 times.

Example 13

Given a rectangular matrix W, size m by n , it is required to form the vector MAX with n elements, the i th element being equal to the element of largest modulus in the i th column of W .

There are two loops: an inner loop to scan a column and find the required element, and an outer loop which arranges for the inner loop to deal with each column in turn. A quantity t has been introduced in the inner loop to make the logic clear; it holds the element of largest modulus found so far during the scan of one column, and at the end of the scan the number t is transferred to the appropriate position in the vector MAX.

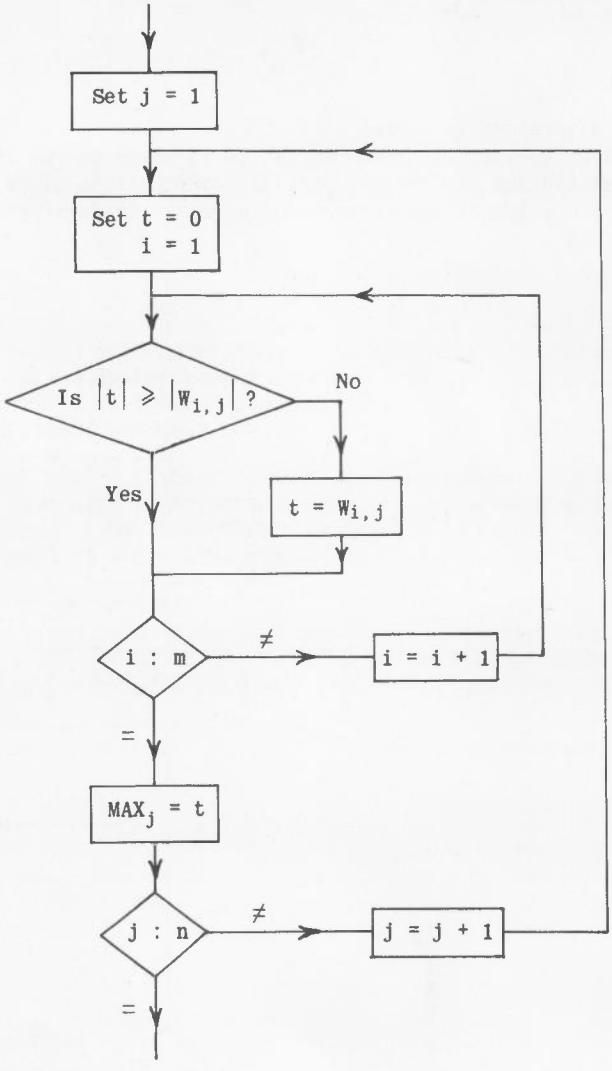


Fig. 12

Note that the logic of the flow of control does not break down if W has only one row or one column, or if all elements in one column are zero.

2.5.7 Subroutines. In large programs, it may be necessary to perform the same section of computation at various points; writing the required instructions at length at each required point will obviously waste both the programmer's time and machine storage space. It is possible to define the required computing procedure once only as a self-contained section of program, known as a subroutine, subprogram or procedure, and to arrange to call this procedure into use whenever desired. Generally speaking, the calling program will have to provide the subprogram with a list of the quantities on which it is to operate at each occurrence, and the subprogram will return to the calling program with one or more computed results. There are, however, many variations in the way subroutines are written and used, and detailed consideration will be left to later sections.

Example 14

P(Z) and Q(Z) are defined by the equations

$$P = \begin{cases} 1 + \left(\dfrac{2000 - Z}{500}\right)^{\frac{1}{2}} & Z < 2000 \\ 1 & Z \geqslant 2000 \end{cases}$$

and

$$Q = \begin{cases} .0045Z - 0.5 \times 10^{-6}Z^2 & Z < 1000 \\ .0035Z + 0.5 & Z \geqslant 1000 \end{cases}$$

A self-contained subroutine may be written which evaluates P and Q, given Z. Then if P and Q are required at several points in one program, this subroutine may be called into use at each point to return to the main program the values of P and Q as determined from the current values of Z.

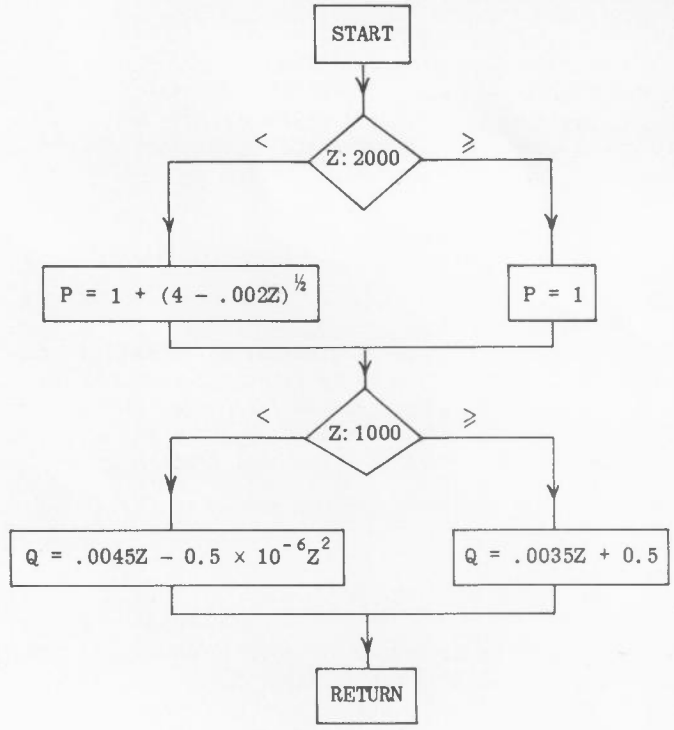


Fig. 13

2.6 Integers and real variables

Although the user of an automatic programming language is not generally concerned with the method of storing a number inside the computer, the fact that it is represented by a group of binary digits of finite length leads to certain restrictions on size and precision which must be understood.

Two types or modes of numbers are considered here - integer and real, sometimes referred to as fixed-point and floating-point respectively. Integers are represented in the computer exactly; they can take on only integral values, and arithmetic operations on integers lead to integer answers. The maximum size of integers allowed depends on the machine and language being used, but for Atlas and Orion is in excess of 10^{10}.

The majority of numbers used in computation will be stored in real form. These are represented in a manner similar to the mathematical notation in which a number is treated as a fraction between 0.1 and 1.0 times a power of ten (the mantissa and exponent respectively). The mantissa is restricted to a precision of about 11 decimal digits, and the exponent has sufficient range to represent numbers much larger than those which will normally occur. Floating-point arithmetic as carried out by the computer automatically handles the position of the decimal point.

It should be appreciated that most fractions have neither an exact decimal nor an exact binary representation. For instance, if the decimal representation of $\frac{1}{3}$ to eight significant digits is added to itself 3 times, the answer is .99999999 and not 1.00000000, and a computer performing this operation would not arrive at the answer 1 exactly; for most practical purposes this lack of exact equality is unimportant, the answer being accurate enough. It can be important under some circumstances. In examples 7, 10, 12 and 13 above, for instance, the counters i, j involved would be stored as integers. In example 12, the program would be theoretically quite satisfactory if the tests "Is j = 11?" and "Is i = 61?" were replaced by "Is t = .15?" and "Is d = 4.00?". But t and d would be real

variables, and the increments .01 and .05 are not exact binary fractions. Consequently t would never exactly equal .15 and the program would repeat the inner loop indefinitely.

As another illustration, the equality $ad - bc = 0$ in example 2 is unlikely to be satisfied exactly, and in practice the test would probably be replaced by something like "Is $|D| < 10^{-7}$?".

Exercises 2

Draw block diagrams for the following:

1. Part of a program to form from the two-dimensional array Z (size p by q) the array S defined by

$$S_{i,j} = \sum_{k=j}^{q} Z_{i,k} \cdot \quad (i = 1,2, \ldots, p \; ; \; j = 1,2, \ldots, q)$$

2. A program to read the number $b(\geq 1)$ and calculate and print out successively the first seven positive roots of the equation

$$b \cos x = 1 - 3e^{-x}$$

correct to 6 decimal places. The program is to repeat this calculation for other values of b, until a value $b = 0$ is read indicating the end of the problem.

Hints: Determine the approximate location of the roots.
Let the program make a first guess at each root, and refine this guess using Newton's method.

3. A subroutine to perform linear interpolation as follows.

A graph $y = f(x)$ is represented by storing a vector of ordinates $x_1, x_2, \ldots, x_n$ in ascending monotonic order and a vector $y_1, y_2, \ldots, y_n$ of corresponding values of $f(x)$. Given a value of x, the subroutine is to calculate a corresponding y by searching the list of ordinates and interpolating linearly between the two values y_{i-1} and y_i where $x_{i-1} \leq x \leq x_i$. If $x < x_1$ or $x > x_n$ the program is to go to an error routine.

3. FORTRAN ARITHMETIC

3.1 Statements

A FORTRAN program or self-contained section of program may be termed a routine, and takes the form of a series of statements having a general resemblance to algebraic formulae and English sentences. Some of these call for the execution of particular steps in the computation and are termed *executable statements*; others are non-executable and declare certain information about the quantities and procedures involved in the program. These are termed *declarations* and will frequently be placed at the head of a routine.

3.2 Assignment of values

A simple executable statement is

```
A = 4.5 + X
```

This is an example of an arithmetic *replacement statement*, having the meaning "Assign to the variable called A a value equal to 4.5 plus the value of the variable called X". Before the statement is executed, X must already possess a value (probably assigned by some previous statement); on the other hand, A may be undefined until the execution of this statement. Thereafter, A will hold its value until later execution of some statement replaces it by a new value. The value of X is unchanged after the statement has been obeyed, since no new value has been assigned to it.

Note particularly that the = symbol denotes assignment or replacement rather than equality. The value of the expression on the right-hand side of this symbol is computed and an assignment of this value is then made to the variable on the left-hand side. For instance, the statement

```
I = I + 1
```

has the meaning "Assign to the variable called I a new value equal to its old value plus 1". Other simple examples are:

```
P = -P          equivalent to changing the sign of P
P = Q           which sets P equal to Q
PI = 3.14159    which sets the value of the variable called PI equal to 3.14159
```

Simultaneous or multiple assignment with more than one variable on the left-hand side is allowed. For example, the statement

```
I,J = 2
```

assigns the value 2 to both I and J.

A, X, I, J, PI and so on above are *variables* referred to by names, their actual values being determined during the course of execution of the object program. The numbers 4.5 and 3.14159 on the other hand, are *constants* appearing in explicit numerical form in the source program.

3.3 Identifiers

The names chosen for variables are termed *identifiers*. An identifier is a string of not more than six characters, the first of which is alphabetic and the remainder either alphabetic or numeric. The

alphabetic characters may be any of the 26 upper case letters of the alphabet A to Z. In general, identifiers may be freely chosen, but a few are used for purposes such as naming special functions and these should be avoided.

Examples of acceptable identifiers are

Z XX MASS JOBNO M2K ROOT1 ROOT2

Note that the numbers 1 and 2 are specifically parts of the last two identifiers; they are not subscripts but merely serve to distinguish between two variables for which it is convenient to choose similar names.

One of the functions of the compiler is to assign storage locations for each identifier occurring in a program; actual machine instructions compiled in the object program refer to these storage locations.

3.4 Mode declarations

Arithmetic variables in a FORTRAN program may be in one of two *modes*, INTEGER or REAL. The compiler may be informed of the modes of variables by *declarations* of the form

```
REAL    MASS,   KAPPA
INTEGER TIME,   COUNT,  A200
```

These declarations are made at the beginning of a routine and apply throughout. A variable whose mode is not declared in this way is assumed to be of mode INTEGER if its identifier begins with I,J,K,L,M or N and of mode REAL if its identifier begins with any other letter.

For instance, identifiers such as I, JOBNO, ICOUNT denote variables of mode INTEGER and identifiers such as A, THETA, P2 denote variables of mode REAL without specific declaration. (Two other types of arithmetic variable are possible, namely COMPLEX and DOUBLE PRECISION. These will not be further discussed in this primer.) Declarations serve to provide the compiler with necessary information; they are not statements which are acted upon when the compiled object program is actually obeyed.

3.4.1 Precision and size. Quantities of mode INTEGER may assume values within the range:

for Atlas $-8^{12} \leqslant I < 8^{12}$ ($-10^{10} < I < 10^{10}$ approximately)

for Orion $-2^{47} \leqslant I < 2^{47}$ ($-10^{14} < I < 10^{14}$ approximately)

Quantities of mode REAL may be zero or have absolute magnitude lying between approximately:

for Atlas 10^{-115} and 10^{115}

for Orion 10^{-38} and 10^{38}

The precision of REAL numbers is about 11 significant decimal digits.

3.5 Constants

An unsigned constant may be written in a FORTRAN program as a string of decimal digits, with possibly a decimal point before all digits, between two digits or after all digits, and possibly followed by an exponent part consisting of the character E followed by an optional + or - sign and a decimal integer. The exponent part gives the power of 10 to be applied as a scale factor.

Only if there is neither an exponent part nor a decimal point will the constant be taken as an INTEGER; it will otherwise be stored in REAL form.

Examples

INTEGER constants	0	
	12	
	54796	
REAL constants	0.	
	2.0	
	.5E+6	having the value 0.5×10^{6}
	0.4516E1	having the value 0.4516×10^{1}
	45.16E-1	having the value 45.16×10^{-1}

3.6 Arithmetic expressions

The right-hand sides of the replacement statements in paragraph 3.2 are simple examples of arithmetic expressions. FORTRAN provides for five *arithmetic operations*, denoted by the following symbols.

exponentiation	**	e.g. A ** 2 means A^2
multiplication	*	e.g. A * B means A multiplied by B
division	/	e.g. A/B means A divided by B
addition	+	
subtraction	-	

Variables and constants may be combined using these symbols to form *arithmetic expressions*, using parentheses where necessary to group quantities together to indicate the order of evaluation, as in ordinary mathematical notation. The following simple rules must be observed to avoid ambiguities.

(a) The multiplication symbol must be used when multiplication is intended. (For instance, the compiler will assume AB to be a single variable called AB and not the product of two variables A and B .)

(b) Two operation symbols must not appear in sequence. (For instance, one must write A * (-B) or -A * B and not A * -B .)

(c) The *hierarchy of operations* within an expression is as follows:

exponentiation

multiplication and division

addition and subtraction

That is, all exponentiations are performed first, then all multiplications and divisions and then all additions and subtractions. Parentheses are used to override this natural order. A set of constituents linked by operation symbols at one level of this hierarchy is termed a *segment*; the segments of an expression are evaluated one by one in the correct order.

Examples

A + B*C + D, meaning the same as A + (B*C) + D , is an expression with two segments. The segment B*C (= B1, say) is evaluated first and then the segment A + B1 + D .

(A + B)*(C + D) is an expression with three segments. (A + B) and (C + D) are evaluated first and then the product is evaluated.

(A + B)**4 meaning $(A + B)^4$ and A + B**4 meaning $A + B^4$ are expressions each consisting of two segments.

(d) Within one segment, the meaning of a sequence of operations is that obtained by evaluation from left to right.

Examples

A/B*C means the same as (A/B)*C

A/B/C means the same as (A/B)/C and may be written alternatively as A/(B*C)

A - B - C means the same as (A - B) - C

(e) A sequence of exponentiations is an ill-defined notation; the compilers will indicate a trivial error, but apply rule (d). However, (A**B)**C is better written as A**(B*C) and parentheses must be used if A**(B**C) is required. Note that a REAL base may be raised to a non-integer power (provided that the value of the base is never negative). Thus A**B, where A and B are of mode REAL, is a legal expression.

3.7 Mode of expressions

Generally all the constituents of an expression will be of the same mode. If all are REAL, the compiler will produce instructions to evaluate the expression using floating point arithmetic throughout and the complete expression is of mode REAL. Similarly if all constituents are INTEGER, then integer arithmetic is used throughout and the complete expression is of mode INTEGER.

Note that *integer division* is rounded towards zero. For instance, I/J will have the value zero if I = ±3 and J = ±4 .

If some constituents of an expression are REAL and some INTEGER, then the value depends upon the mode of its segments. If all constituents of a segment are INTEGER then the segment is treated as an INTEGER sub-expression and its mode is INTEGER. Otherwise any INTEGER constituents are converted to REAL form except when a REAL constituent is raised to an INTEGER power, and the segment treated as a REAL segment. Typical uses of mixed mode occur in the following expressions, where A,B,C are REAL variables.

A + B - 2	is a REAL segment. The integer 2 is converted to REAL form before evaluating the segment.
B**2 - 4*A*C	is a valid expression with three segments. The integer 4 is converted to REAL form before evaluating the product 4*A*C.
(A+2) * (B+3)	The integers 2 and 3 are converted to REAL form before evaluating the segments (A+2) and (B+3).

Error is most likely to occur in a situation such as the following. If it is required to form the value $A - \frac{5}{3}$, the expression A - 5/3 will give the wrong answer, because 5/3 is an INTEGER segment having the value 1 when rounded towards zero. The expression may conveniently be written as A - 5.0/3.0 or A - 5/3. or even A - 5/3 * 1.0

3.8 Functions

Certain common mathematical functions are provided in the FORTRAN language, and are given standard names. These are listed at the end of this chapter. Typical functions are square root (SQRTF), natural

logarithm (LOGF) and sine (SINF). (Functions may also be defined by the programmer; rules for constructing these are given in section 8.)

The name of a function it is desired to use is followed, in parentheses, by an expression whose value is to be determined and used as the actual argument of the function. Certain functions operate on two or more arguments: the expressions supplied as actual arguments are then listed separated by commas. A function computes a single value and may be used as a constituent of an expression wherever a variable or constant may appear. An argument of a function, being an expression, may well involve references to other functions or to the same function.

Some examples of expressions involving functions follow.

```
1 + LOGF(A + B)

(-B + SQRTF(B**2 - 4*A*C))/(2*A)

LOGF(SINF(X) + COSF(X))

SQRTF(SQRTF(Y))

AMAXF(2*A, 3*B, C/2)
```

3.9 Arithmetic statements

As indicated in paragraph 3.2, an *arithmetic replacement statement* is an instruction to compute the value of the arithmetic expression written on the right-hand side of the symbol = , and to assign the value to the variable (or variables) on the left-hand side.

If the mode of the expression is INTEGER, but the variable is of mode REAL, the necessary conversion is made automatically. Thus the statement A = I is treated as A = FLOATF(I) .

The programmer is advised to avoid statements requiring the assignment of the value of a REAL expression to an INTEGER variable, unless the form of truncation required is specified by using one of the standard functions INTF, NINTF or IFIXF. Thus the statement I = NINTF(A) is quite acceptable and unambiguous. (The reference manuals give details of the behaviour of the compiler in ambiguous situations created by statements such as I = A.)

In Orion FORTRAN the variables appearing on the left-hand side of a multiple replacement statement must be all of the same mode, either all REAL or all INTEGER. This restriction does not apply to Atlas FORTRAN, subject to the qualifications of the previous paragraph. Thus the statement SUM, I = 0 is valid on Atlas but not on Orion.

3.10 Lists of standard functions

The tables overleaf list the principal standard functions.

FUNCTIONS WITH ONE ARGUMENT

X denotes a REAL argument; I denotes an INTEGER argument.

Name	Mode of result	Mode of argument	Definition	Notes and examples
SQRTF	R	R	Positive square root of X	
SINF	R	R	Sine of X	Argument in radians
COSF	R	R	Cosine of X	" " "
TANF	R	R	Tangent of X	" " "
LOGF	R	R	Natural logarithm of X	
EXPF	R	R	Exponential of X	
TANHF	R	R	Hyperbolic tangent of X	Not available in Orion FORTRAN
ASINF	R	R	Arcsine of X i.e. $\sin^{-1}X$	Result in radians $-\pi/2 \leqslant$ result $\leqslant \pi/2$
ACOSF	R	R	Arccosine of X i.e. $\cos^{-1}X$	Result in radians $0 \leqslant$ result $\leqslant \pi$
ATANF	R	R	Arctangent of X i.e. $\tan^{-1}X$	Result in radians $-\pi/2 \leqslant$ result $\leqslant \pi/2$
FLOATF	R	I	Convert I to floating form (real form)	
IABSF	I	I	Absolute value of I i.e. $\lvert I \rvert$	
ABSF	R	R	Absolute value of X i.e. $\lvert X \rvert$	
INTF	I	R	Largest integer $\leqslant$ X	Mathematical integral part e.g. INTF(3.7) = 3
AINTF	R	R	Largest integral real value $\leqslant$ X	Equivalent to FLOATF(INTF(X)) INTF(-3.7) = -4
IFIXF	I	R	Sign of X times largest integer $\leqslant \lvert X \rvert$	Equivalent to INTF for positive X e.g. IFIXF(3.7) = 3
FIXF	R	R	Sign of X times largest integral real value $\leqslant \lvert X \rvert$	Equivalent to FLOATF(IFIXF(X)) IFIXF(-3.7) = -3
NINTF	I	R	Nearest integer to X	e.g. NINTF(3.7) = 4 NINTF(3.2) = 3 NINTF(-3.7) = -4

FUNCTIONS WITH TWO ARGUMENTS

X1, X2 denote two REAL arguments; I1, I2 denote two INTEGER arguments

Name	Mode of result	Mode of arguments	Definition	Notes and examples
ISIGNF	I	I	Sign of I2 times IABSF(I1)	e.g. ISIGNF(10, -4) = -10 ISIGNF(-10, -4) = -10
SIGNF	R	R	Sign of X2 times ABSF(X1)	
IDIMF	I	I	I1 - I2 if I1 > I2, otherwise 0	e.g. IDIMF (5, 3) = 2 IDIMF(3,5) = 0
DIMF	R	R	X1 - X2 if X1 > X2, otherwise 0	IDIMF(-7, -8) = 1
MODF	I	I	I1 - I2 * IFIXF(I1/I2)	e.g. MODF(7, 3) = 1 MODF(-7, 3) = -1
AMODF	R	R	X1 - X2 * FIXF(X1/X2)	

FUNCTIONS WITH TWO OR MORE ARGUMENTS

X1, X2, X3 ..... denote REAL arguments; I1, I2, I3 .... denote INTEGER arguments

MAXF	I	I	Largest of I1, I2, I3 ...	i.e. algebraically largest
AMAXF	R	R	Largest of X1, X2, X3 ...	" " "
MINF	I	I	Least of I1, I2, I3 ...	i.e. algebraically least
AMINF	R	R	Least of X1, X2, X3 ...	" " "

Exercises 3

1. Which of the following are unacceptable identifiers? If no specific declarations are made, what is the mode of each acceptable identifier?

B(1)	TEMP	JCOUNT
A2.4	3K	△
WEIGHT	T-STEP	U2
BBB	DISTANCE	SUM

2. Express the following numbers without writing an exponent part.

12E2 1.473E + 2 0.0321E - 6 125.67E4

3. Write FORTRAN expressions corresponding to the following expressions written in conventional mathematical notation.

(i) $\dfrac{A + 4}{2B}$

(ii) $\dfrac{1}{7}\left(B + \dfrac{1}{7}\right)^{\frac{1}{2}}$

(iii) $3(A + B)$

(iv) $(B^2 + 2D \times E)^2$

(v) A^{J+1}

(vi) $\dfrac{(A + B)\ (C + D)}{E + F}$

(vii) $AX + BX^2 + \dfrac{CX^3}{D}$

(viii) $\left(\dfrac{A}{A + 1.5}\right)^{3.5}$

(ix) $\log\left(\dfrac{A}{B(C + D)}\right)^{\frac{1}{2}}$

(x) $\sqrt{\dfrac{A + \dfrac{B}{C + 1}}{D}}$

4. Write arithmetic statements to perform the following steps of computation. Use the names of the variables in the formulae as identifiers, changing to upper case letters where necessary.

(i) $SUM = \dfrac{1}{4} N^2 (N + 1)^2$ (N is an integer)

(ii) $q = \dfrac{27\ a^2d - 9abc + 2b^3}{27a^3}$

(iii) $T = \dfrac{b - c}{b + c} \cot \dfrac{A}{2}$

(iv) $T = \sqrt{\dfrac{(s-b)\ (s-c)}{s(s-a)}}$

(v) $S = \log[1 + \sqrt{(1+x^2)}] - \log x$

(vi) $INT = \frac{1}{2} \sec x \tan x + \frac{1}{2} \log (\sec x + \tan x)$

(vii) $Area = \pi(R + S) \sqrt{h^2 + (R-S)^2}$

(viii) $CG = \dfrac{h}{4} \dfrac{(4R - h)}{(3R - h)}$

(ix) $z = y = e^x \cos 2x$

(x) $z = \log\left|\dfrac{1}{\tan y}\right|$

(xi) $u = \sqrt{\dfrac{x}{3} \tan 3x}$

5. Indicate the error in each of the following would-be arithmetic statements.

(i) A = B = C**2 + 1

(ii) A = B / - 2*C

(iii) A = (B+1) (C-1)

(iv) A = 4 LOGF (A + 0.2)

(v) A = SQRTF (B + 2*(C+D)

(vi) A+2 = (B+C) / (-B+C)

(vii) A = 4/3 * π * R**3

6. A = 4.0, B = -6.0, C = 2.0 and D = 5.0 are real variables. I = -4, J = 1 and K = 3 are integer variables.

Evaluate the results of the following statements, where X, Y are real variables and N is an integer variable.

(i) X = (A**1.5 + 2) / B * D

(ii) X = C* (D-C) / (2*A)

(iii) N = I + J/K + I/K

(iv) N = K + 1 / (J-3)

(v) Y, X = -B + C**(K+1)

(vi) X = (B+C/I)** 2

(vii) N = INTF((B + 2 * K/I)/D)

4. THE WRITTEN FORM OF A FORTRAN PROGRAM

4.1 Standard programming form

The statements of a FORTRAN program are written on a standard programming form: the program is presented to the computer in some representation of its written form, for example as punched cards or paper tape, for translation by the compiler into the object program which will actually carry out the processes specified in the source program. Besides performing the translation, the compiler will print out a listing of the source program exactly as input; this considerably eases the problem of documentation.

A typical programming form is shown in Figure 14. Each line of the form accommodates 80 character positions, corresponding to the columns of a punched card. (The meaning of some of the statements in the example written on the form will be explained in later sections.)

FORTRAN CODING FORM			PROGRAM TITLE: Fig.14 : EXAMPLE	DATE:	SHEET:
				AUTHOR:	
STATEMENT No. 1 2 5		6	7 FORTRAN STATEMENT 72	73 LABEL 80	
π			PROGRAM FOR EXAMPLE 6, SECTION 2		
	1		READ 100,A,B,C		
	100		FORMAT (3F10.0)		
			IF (A),EXIT, π DETECT END OF DATA		
			A1 = SQRTF (B**2 + C**2 - 2*B*C*COSF(A))		
			DELTA = 0.5*B*C*SINF(A)		
			PRINT 101,A1,DELTA		
	101		FORMAT (2E20.8)		
			GO TO 1		
			END		

Fig. 14

4.2 Statements

Each statement is started on a new line, and may be continued on as many lines as necessary. It is confined to positions 7 to 72 of each line. In the first (or only) line of any statement position 6 must be zero or blank; continuation lines are indicated by writing in position 6 some character other than zero or blank. (They may, for instance, be serially numbered in position 6.)

The nature of a statement must be completely defined by its first line, which places slight restrictions on the use of continuation lines. For instance, in a multiple replacement statement the = symbol must appear on the first line.

4.3 Statement numbers

Some statements in a FORTRAN program require labelling with a statement number, which may have any value up to 99999; it is written anywhere in positions 1 to 5 of the first (or only) line of the statement. No particular significance is attached to the order or size of statement numbers; they provide simply a convenient means of labelling.

4.4 Comments

Comments may be used to improve the intelligibility of a program to anyone who may have to read it. They are ignored during compilation and make no contribution to the resulting object program. A comment may be a complete line introduced by the character π (or C) in position 1, or that part of a line including and following the character π in any of the positions 7 to 72. Comments will normally be written between statements, but the occurrence of comment does not in fact indicate the end of a statement, which may continue on the following line.

4.5 Blanks

Blank spaces are ignored by the compiler (except in position 6 and in certain situations using constant TEXT which will be mentioned later); they may be freely used to improve legibility by insetting lines, leaving blank lines and separating symbols and names, at the programmer's discretion.

4.6 Serial numbers

Positions 73 to 80 of each line are ignored by the compiler and may be used for labelling routines and serially numbering lines as desired.

4.7 Characters

The complete set of characters necessary to write the full FORTRAN source language is:-

ABCDEFGHIJKLMNOPQRSTUVWXYZ

0123456789

().,=*/+-

To avoid ambiguities and punching errors care should be taken to write legibly and to distinguish between similar characters such as 1 and I, 2 and Z, 5 and S and the letter O and zero. It is suggested that letter O be written as $\bar{O}$ and letter Z as $\bar{Z}$. Every symbol required by the prescribed rules for writing particular types of statement must be inserted, including commas and other separating symbols.

5. INPUT, OUTPUT AND FORMAT STATEMENTS

In order to read in data and output results, a FORTRAN program will include certain *input* and *output statements* such as READ and PRINT. These are executable statements which will call for data to be read in or output via some peripheral device. It should be appreciated that apart from those constants which are peculiar to a program and are the same whenever it is executed, the data on which a program operates will vary from one execution to another and so cannot be "built in" to the source program by writing constants in statements. The FORTRAN language provides very comprehensive and flexible facilities for dealing with data in a variety of forms presented on any of the peripheral devices. This section introduces the simpler uses of three statements only, namely READ, PRINT and PUNCH.

5.1 Documents

A program or a set of data recorded on one particular input or output medium may be referred to as a *document*. On both Atlas and Orion use of the statement PRINT will produce a document consisting of printed results from the line printer. Use of the statement PUNCH will normally produce results as a pack of punched cards, on Atlas, or as punched paper tape, on Orion. The term *system output documents* is frequently used to describe these media. The statement READ, used in its simplest form, makes no reference to any input device; it is then assumed to refer to the *system input document*, that is the data is assumed to be found on the same input device as that on which the program is presented to the computer, which will normally be a punched card reader on Atlas or a punched paper tape reader on Orion. For convenience further description will be confined to cards.

The operating systems on both Atlas and Orion are very flexible, and variations in the use of peripheral devices are quite simply catered for; these are usually matters of operating convenience and do not affect the programmer. The interested reader should consult the reference manuals for details, and for information regarding the use of magnetic tape and drums to provide large amounts of working storage.

5.1.1 Records. Documents are made up of *records*; one record may be, for instance, a punched card (length of record 80 characters) or a line of printed matter (length of record 120 characters, preceded by a special carriage control character which is not printed) or the equivalent of one line of printed matter on paper tape. Information about the layout of items within records is provided by *FORMAT statements*. Such information includes the number of values in one record, whether they are integer or real values, the number of characters occupied by each value, whether a floating point representation with an exponent is used, whether a decimal point is explicitly included or its position is assumed, and so on. FORMAT statements are labelled with statement numbers, and an input/output statement must refer to an associated FORMAT statement by its number.

5.2 Output Formats

```
PUNCH 137, A, N, TIME, LINE
137 FORMAT (E16.8, I4, F8.3, I2)
```

Any input/output statement starts a new record. In this example, the PUNCH statement calls for the current values of the variables called A, N, TIME and LINE to be punched on a new card, according to the format given in the statement which has been numbered 137. Note that the PUNCH statement quotes the FORMAT statement number first, followed by an *output list* of variable names separated by commas.

The punching starts in the first column of the card and proceeds from left to right in consecutive columns. The values are punched in the order obtained by scanning the output list from left to right - in this example in the order A, N, TIME, LINE. This scanning process applies also to the FORMAT statement; each time a new value from the output list is to be punched, the next *field specification* in the FORMAT statement is associated with it. Thus the first specification E16.8 is associated with the value of A, the second specification I4 with N and so on. Note that the list of field specifications, separated by commas, is enclosed in parentheses. The FORMAT statement is a declaration: it is frequently but not necessarily written immediately following the input/output statement which refers to it.

The meaning of the field specifications may be illustrated using the above example; by the term *field width* is meant the number of consecutive character positions to be occupied by a number.

(a) The E-type specification indicates that the value of A is to be punched in a floating form with a mantissa and an exponent. The figure 16 indicates that the field width is to be 16 characters and the figure 8 indicates that there are to be 8 places to the right of the decimal point in the mantissa.

Typical output: (on Atlas)

value of A	actual punching
0.5	bb5.00000000E-01
-12.12345678	b-1.21234568Eb01
.00753261234	bb7.53261234E-03

(The letter b has been used to denote those positions which will contain a blank character).
Features of E - type specification:

(i) The characters are right-justified within the field, that is, they are punched as far to the right as possible.
(ii) The number is in a standardised form. On Atlas the mantissa lies in the range $1 \leqslant \text{mantissa} < 10$ as indicated in the example above; on Orion it lies in the range $0.1 \leqslant \text{mantissa} < 1$.
(iii) Only minus signs are punched.
(iv) The characters E and decimal point are explicitly punched, and must be included in the count of characters in the field width.
(v) Numbers are correctly rounded off.

(b) The I - type specification indicates that the value of N is to be punched as an integer. The field width is to be 4 characters.

Typical output:

value of N	actual punching
2	bbb2
-40	b-40

Features of I - type specification:

(i) The characters are right - justified within the field.
(ii) Only minus signs are punched.

(c) The F - type specification indicates that the value of TIME, a real variable, is to be punched in a form without an exponent. The field width is to be 8 characters, and there are to be 3 places to the right of the decimal point.

Typical output:

value of TIME	actual punching
14.5	bb14.500
-373.4567	-373.457

Features of F - type specification:

(i) The characters are right-justified within the field.
(ii) Only minus signs are punched.
(iii) The decimal point is explicitly punched and must be included in the count of characters.
(iv) Numbers are correctly rounded off.

5.2.1 Incorrect field widths. If the value of a number to be output is such that it cannot be represented within the field width given in the FORMAT specification, any adjacent field to the left is not interfered with. On Atlas the value will be partially output with its most significant character or characters missing. For example, using an I4 specification a value of -1234 will be output as 1234, losing the minus sign, and a value of 56789 will be output as 6789. On Orion no digits will be printed, but the field will be filled with + or - signs according to the sign of the number.

5.2.2 Printed output. In the above example the numbers are closely packed on a punched card. Legibility of *printed* results is improved by extending field widths so as to include blank spaces before each number. The upper limit of 121 characters, of which the first is not printed, must be remembered.

```
      PRINT 12, I, A, B, X
12 FORMAT (I6, E20.6, E20.6, F12.3)
```

A typical line printed by this statement is as follows. (In this and other examples of printed output in this section and in section 10 a vertical bar will be used to denote the start of a printed line, which will be aligned with the left hand margin.)

```
| 109      -2.456167E 02       2.571663E-01      247.158
```

5.3 Repetition of field specifications

A useful "shorthand" is available which may be illustrated by rewriting the above FORMAT statement as

```
12 FORMAT (I6, 2E20.6, F12.3)
```

The integer 2 signifies that the specification E20.6 is to be used twice in succession during the scan of the FORMAT statement before passing to the next specification F12.3.

A group of specifications may be repeated by enclosing the group in parentheses and preceding it by an integer: such a repeated group may be part of a longer list of specifications.

```
      PRINT 10, K, I, A, J, B
10 FORMAT (I5, 2 (I8, E20.8))
```

Typical output:

```
| 100      12     -9.21376254E 02      132      1.57399897E-04
```

5.4 Input formats

Input formats are essentially the same as those used for output. However, whereas the field specifications are rigidly adhered to on output considerable deviation is allowed on input.

The rules may be summarised as follows:-

(a) The field width must be adhered to.
(b) Embedded blanks are illegal in Atlas FORTRAN and treated as zeros in Orion FORTRAN. Consequently integers and the exponents of floating form numbers must be right-adjusted within the field.
(c) Plus signs may be punched. If there is no sign a number is taken as positive.
(d) A floating form number is recognised by the presence of an exponent part which must be separated from the mantissa by either the character E, or by a sign, or by E and a sign.
(e) Numbers in floating form must be input using an E - type specification; numbers without an exponent must be input using an F - type specification.
(f) The decimal point need not be punched, in which case the field specification gives the number of decimal places. If the decimal point is punched, its position overrides that given in the field specification.

Examples

Field specification	Input data	Value
I6	bbb123	+123
	bb+123	+123
	-56789	-56789
E12.6	bbb12345E+02	+1.2345
	+123.456Eb01	+1234.56
	bbbb-1234-02	-.00001234

Field specification	Input data	Value
F10.4	bbbb456789	+45.6789
	bb+4.56789	+4.56789
	bbbbbbb470	+.0470

5.5 Simple programs

For general use when the size of input data and results is not known in advance, specifications such as F10.0 for input, and E20.8 for printed output are recommended. The data is punched with an explicit decimal point to override the F specification, and the results will be correctly printed regardless of the actual size of the numbers involved, with eight or nine significant figures and adequate space to provide legibility.

5.5.1 G-Type specification. In Atlas FORTRAN only a useful additional field specification is provided, namely the G-type field specification, which has the form Gw where w is an integer specifying the field width. The characteristics of this specification make it particularly suitable for general output; it calls for output of either REAL or INTEGER numbers within a field width of w characters. Six character positions in the field are required for the decimal point which is always included, a minus sign where necessary and an exponent part when appropriate. Consequently w-6 characters remain, and any number will be output with w-6 significant figures (up to a maximum of 12), the exponent part being required only if the number is of such a magnitude that a decimal point cannot otherwise appear or is less than 1 so that loss of significant figures otherwise occurs.

Typical output of numbers printed according to the specification G13 is:

value	actual printing
463	463.0000
-4.6372156	-4.637216
0.463721569	4.637216E-01
-463721569	-4.637216E 08
4637215	4637215.

So long as an exponent part is not required the decimal point is allowed to float within the field. Otherwise the number is output in standard floating form. If w is greater than 18, 12 significant figures are printed and preceding blanks inserted.

Numbers input from a G-type field may be with or without an exponent, and are taken to be REAL. The number of decimal places is assumed zero if an explicit decimal point is not included in the data.

5.5.2 Format-free input

Also in Atlas FORTRAN only, a format statement may be omitted altogether. On input the data is then format-free, numbers being separated by one or more blanks; all will be treated as REAL. When required, decimal points must of course be explicitly included. (If no format statement is given on output, then

FORMAT (6G20)

is assumed.)

5.6 EXIT and END

The exercises at the end of this section may be made complete programs by finishing with the two statements

GO TO EXIT (Atlas) or CALL EXIT (Atlas or Orion)

and END

The first of these is dealt with in the next section. The END statement does not form part of the program itself, but is a signal to the compiler that the physical end of all the program statements has been reached.

Exercises 5

Write complete programs to carry out the following simple calculations. The variables involved are real variables unless otherwise stated. Make suitable choices of identifiers and include any necessary mode declarations.

1. Read the values of C, E, f, L and R from one card; print out on one line these values and the value of

$$I = \frac{E}{\sqrt{R^2 + \left(2\pi fL - \frac{1}{2\pi fC}\right)^2}}$$

No input quantity will be given to more than 6 significant figures.

2. Read a positive integer p ($< 10^3$) and print out on one line the values of

$$p, \sum_{n=1}^{p} n = \tfrac{1}{2} p(p+1) \text{ and } \sum_{n=1}^{p} n^2 = \frac{1}{6} p(p+1)(2p+1)$$

3. Read the lengths b and c of the two sides of a plane triangle and the included angle A in radians. Punch on one card a, b and c, and on a second card Δ, where

$$a = \sqrt{(b^2 + c^2 - 2bc \cos A)}$$

and

$$2\Delta = bc \sin A.$$

4. Read the three angles A, B, and C of a spherical triangle and compute the three 'sides' a, b and c from the formula

$$\tan \tfrac{1}{2}a = \sqrt{\frac{\sin \tfrac{1}{2}E \sin (A - \tfrac{1}{2}E)}{\sin (B - \tfrac{1}{2}E) \sin (C - \tfrac{1}{2}E)}}$$

and the two similar formulae for $\tan \tfrac{1}{2}b$ and $\tan \tfrac{1}{2}c$, where $E = A + B + C - \pi$ (radians).

A, B and C are given in degrees to three decimal places and are to be read from one card. a, b, and c are to be printed on one line, also in degrees to three decimal places.

6. CONTROL STATEMENTS

6.1 Transfer of control

Execution of a FORTRAN routine starts with its first executable statement, and thereafter executable statements are taken in the order in which they are written until a specific break in this order is called for. Just as input/output statements may refer to FORMAT statements by their numbers, so various *control statements* may specify their *successors* out of normal sequence by referring to statement numbers.

The simplest control statement is the unconditional GO TO. For example, the statement GO TO 12 when executed causes control to be transferred to the statement numbered 12.

6.2 Arithmetic IF statements

The most useful control statement is the *arithmetic IF statement*, which gives a conditional transfer of control: that is, a transfer which depends upon a computed result. This statement takes the general form

IF (A) n_1, n_2, n_3

A may be any arithmetic expression (a single variable being merely a special case of an expression) and is enclosed in parentheses. n_1, n_2 and n_3 are the three possible successors to this statement, not necessarily all different; control is transferred to statement n_1 if $A < 0$, to statement n_2 if $A = 0$ and to statement n_3 if $A > 0$. By convention, the next executable statement may be specified as a successor, if it has no statement number associated with it, by simply omitting n_1, n_2 or n_3 as appropriate: the two commas must nevertheless still appear.

Examples

(a) The statement

```
IF (X) 12, 13, 16
```

transfers control to statement 12 if $X < 0$, to statement 13 if $X = 0$ and to statement 16 if $X > 0$.

(b) The statement

```
IF (B**2 - 4*A*C) 100,,
```

transfers control to statement 100 if $B^2 < 4AC$ and to the next executable statement if $B^2 \geqslant 4AC$.

(c) The statement

```
IF (THETA - 0.5),,10
```

transfers control to the next executable statement if THETA $\leqslant 0.5$ and to statement 10 if THETA > 0.5.

(d) The following section of program is one possible way of solving example 9 of section 2. It illustrates the use of an arithmetic IF statement and an unconditional GO TO to program a loop.

```
      E = 2
      TERM = 1
      N = 2
   21 TERM = TERM/N
      E = E + TERM
      IF (TERM - 1E-8) 20, 20,
      N = N + 1
      GO TO 21
   20 (next statement of program)
      ..........
```

6.3 Logical IF statements

Two arithmetic expressions A and B may be compared by using a *relation* which may have one of six forms.

A.GT.B meaning $A > B$

A.GE.B meaning $A \geqslant B$

A.LE.B meaning $A \leqslant B$

A.LT.B meaning $A < B$

A.EQ.B meaning $A = B$

A.NE.B meaning $A \neq B$

A relation is said to have a logical value, either .TRUE. or .FALSE.

A *logical IF statement* takes the general form

IF (A) n_1, n_2

where A is a logical expression. (See section 9) In particular, A may be any relation; control is transferred to statement n_1 if A has the value .TRUE. and to statement n_2 if A has the value .FALSE. The next executable statement may be specified as a successor by omitting either n_1 or n_2 as with arithmetic IF statements.

Examples

(a)
```
IF (A.EQ.0) 9, 15
```
transfers control to statement 9 if A = 0 and to statement 15 otherwise.

(b)
```
IF ((B**2).LE.4*A*C) 100,
```
transfers control to statement 100 if $B^2 \leqslant 4AC$ and to the next executable statement if $B^2 > 4AC$. In this case parentheses are required round the expression B**2 to avoid ambiguity which would otherwise arise from a decimal point following the integer 2.

6.4 Named successors

So far in this section a successor of a control statement has been identified by a number. Certain control statements may also be written with the *name* of a successor in place of a statement number under two circumstances.

Firstly, the successor may be a named routine with no arguments and from which no return to this routine is expected. This is allowed only in Atlas FORTRAN. Routines are dealt with more fully in section 8. At this stage it is sufficient to point out that EXIT is the name of a standard routine which is entered at the end of any program. Hence statements such as

```
GO TO EXIT
```

used in the previous section, and

```
IF (A) EXIT, 10, 10
```

are meaningful in Atlas FORTRAN. In the second example, control will be transferred to the routine EXIT if A < 0 and otherwise to statement 10.

Secondly, an identifier may be used to denote a variable label and may occur in control statements wherever a statement number may appear; before a jump to such a named successor can be meaningful, a statement number (or a routine name in Atlas FORTRAN) must have been assigned to the variable label by a previously executed *ASSIGN statement*.

Examples

```
ASSIGN 5 TO CNTROL (Atlas or Orion)

ASSIGN (EXIT) TO CNTROL (Atlas only)
```

The presence of the ASSIGN statement in the routine is sufficient to define the identifier as the name of a label. If a statement number (or a routine name) has not been previously assigned to a name before a control statement requires the use of that name during execution, a failure routine is entered and the program halted.

Examples

(a) If the statements
```
ASSIGN 5 TO CNTROL
      :
      :
      :
GO TO CNTROL
```
and later
are executed in that order, then GO TO CNTROL is equivalent to GO TO 5.

(b) If the statements
```
ASSIGN 4 TO LINK
.............
ASSIGN (LINK) TO CNTROL
.............
GO TO CNTROL
```
are executed in that order then GO TO CNTROL is equivalent to GO TO 4.

It is not permitted to use an arithmetic expression in an ASSIGN statement; an assigned GO TO is used to follow various branches of computation in a *preset* manner; (the statement numbers have no numerical significance). For compatibility with other versions of FORTRAN, a list of possible values of the statement numbers which may be assigned may follow the assigned GO TO, thus:

```
GO TO JUMP, (5, 6, 7, 12, 15)
```

6.5 Computed GO TO

To follow different branches according to a computed value, the *computed GO TO statement* may be used. This takes the general form

GO TO $(n_1, n_2, \ldots, n_m)$, I

where $n_1, n_2, \ldots, n_m$ are the possible successors of this statement (either statement numbers or names), and I is an expression whose value determines which successor is taken at any particular execution: I should be an INTEGER expression or variable. If the value of I is 1 then n_1 is the successor, if the value is 2 then n_2 is the successor and so on. The programmer is responsible for ensuring that the values of I which occur during execution lie between 1 and m. The next statement may be named as a successor by omitting any of n_i in the usual way, for instance

```
GO TO (10, CNTROL,, 104, 53),I
```

Example

A variable Y is defined by different formulae in a series of intervals of X as follows.

$$
\begin{aligned}
Y &= -2 && \text{if } X \leqslant -2 \\
&= -1 + \frac{X}{2} && \text{if } -2 \leqslant X \leqslant -1 \\
&= \frac{5}{2}X + X^2 && \text{if } -1 \leqslant X \leqslant 0 \\
&= \frac{5}{2}X && \text{if } 0 \leqslant X \leqslant 1 \\
&= \frac{-5}{4} + 5X - \frac{5}{4}X^2 && \text{if } 1 \leqslant X \leqslant 2 \\
&= \frac{15}{4} && \text{if } X \geqslant 2
\end{aligned}
$$

A section of program to compute the value of Y, given a value of X, could be as follows.

```
    IF (ABSF(X) - 2), 101, 101
    GO TO (102,103,104,105), INTF(X) + 3
102 Y  = -1 + X/2
    GO TO 107
103 Y  =  (2.5 + X)*X
    GO TO 107
104 Y  =  2.5*X
    GO TO 107
105 Y  =  -1.25 + X*(5 - 1.25*X)
    GO TO 107
101 IF (X) 106,,
    Y  =  3.75
    GO TO 107
106 Y  =  -2
107 CONTINUE
```

The first statement checks whether $X \leqslant -2$ or $\geqslant +2$. If so, statement 101 is taken as the successor; this checks whether X is negative or not. If X is negative (and so $X \leqslant -2$) statement 106 is executed to set Y = -2. If X is positive, on the other hand (and so $X \geqslant 2$), the next statement after 101 is executed since no successor is explicitly named, and this sets Y = 3.75. The following statement, namely GO TO 107, is required to skip over the statement 106 in this case.

Reverting to the first statement, if ABSF(X) - 2 is negative, then X lies in the range $-2 < X < 2$ and the next statement is executed and computes INTF(X) + 3. This expression will have one of the four values 1, 2, 3 or 4 corresponding to the four possible ranges $-2 < X < -1$, $-1 \leqslant X < 0$, $0 \leqslant X < 1$ and $1 \leqslant X < 2$. The computed GO TO statement thus directs control to one of four possible statements for computing Y, according to the value of I.

6.6 The dummy statement

The statement CONTINUE is a dummy statement which executes no operation. It may be useful for setting a label as in the above example. (See also section 6.7.1.)

6.7 DO statements

The IF and GO TO statements introduced in the preceding sections are sufficient to control a program loop. Example 9 of section 2 has been illustrated; the following program is a possible solution to example 8 of the same section.

```
      REAL M, M2, K
      READ 20, NLIMIT, M2
   20 FORMAT (I4, F6.3)
      K = LOGF (M2/100)/LOGF (2)
      N = 1
   22 M = 100 * N ** K
      PRINT 21, N, M
   21 FORMAT (I6, F10.3)
      IF (NLIMIT - N) 23, 23,
      N = N + 1
      GO TO 22
   23 CALL EXIT
      END
```

(The identifiers N, NLIMIT, M, M2 and K have been used to denote the variables n, N, m, M and k respectively.)

Control of a loop by a counter or *index*, as in this example, occurs so frequently that two comprehensive statements, the DO statement and the FOR statement, are introduced into the FORTRAN language for this special purpose.

The general form of the DO statement is

DO n I = K_1, K_2, K_3

Here n is a statement number and I is the *index*; K_1, K_2, and K_2 are *indexing parameters*. In broad terms, this statement calls for repeated execution of the group of statements following it, up to and including the statement numbered n (termed the *range* of the DO statement), with the index assuming first the value given by K_1 and then successive values in the sequence $K_1 + K_3$, $K_1 + 2K_3$, $K_1 + 3K_3$, ... as long as the value K_2 is not passed. After the final execution control passes to the first executable statement after the statement numbered n .

As an illustration, the above example may be rewritten using a DO statement as follows.

```
      REAL M, M2,K
      READ 20, NLIMIT, M2
   20 FORMAT (I4, F6.3)
      K = LOGF (M2/100)/LOGF (2)
      DO 22 N = 1, NLIMIT, 1
      M = 100 * N ** K
   22 PRINT 21, N, M
   21 FORMAT (I6, F10.3)
      CALL EXIT
      END
```

6.7.1 Rules for constructing DO loops. The action of a DO statement requires stating more precisely than in the above description.

(a) n must be a statement number; it may not be the name of a label.

(b) The *index* must be an INTEGER unsubscripted variable. The *indexing parameters* should be INTEGER expressions.

(c) If the increment K_3 is 1 then it may be omitted, together with the preceding comma, and the value 1 will be assumed by the compiler. Thus the DO statement in the above example may in fact be written

DO 22 N = 1, NLIMIT

(d) The increment may be positive or negative.

(e) The values of the indexing parameters are worked out once only on first entry to the loop. The loop is always executed at least once with I = K_1. The last execution of the loop is when I has its greatest value $\leqslant K_2$, if $K_3 \geqslant 0$ or when I has its least value $\geqslant K_2$, if $K_3 < 0$. Thus

(i) DO n I = 10, 6, -1
calls for execution of a loop with I taking the values 10, 9, 8, 7, 6 in turn.

(ii) DO n I = 2, 11, 4
calls for execution of a loop with I taking the values 2, 6, 10 in turn.

(f) No statement in the loop should change the value of the index. Statements in the loop may change the values of variables appearing in the indexing parameters, but these have no effect on the number of executions by virtue of (e).

(g) The last statement in the loop must be an executable statement; it must not be, for instance, a mode declaration or a FORMAT statement.

(h) A DO loop which is left after execution with all possible values of the index is said to be satisfied. A loop may also be left before it is satisfied by a transfer of control to some

statement outside the loop. The final statement in the loop may not however be a transfer statement of any type, and the dummy statement CONTINUE, should be used when such a situation would otherwise occur.

An example is provided by rewriting example 9 of section 2 using a DO statement as follows.

```
      E = 2
      TERM = 1
      DO 21 N = 2, 20
      TERM = TERM/N
      E = E + TERM
      IF (TERM - 1E-8) 20, 20, 21
   21 CONTINUE
   20 (next statement of program)
```

The loop will be left when a term has been calculated whose value is $\leqslant 10^{-8}$. Rather than work out precisely how many terms are required, it is sufficient merely to specify a larger number of repetitions of the loop, say 20, than are necessary.

It is recommended that a programmer always uses a separate CONTINUE statement to mark the end of each DO loop, to avoid inadvertently contravening the above rule and to make the program structure clearer.

(i) A jump into the range of a DO statement is never permitted. (In this context, however, a call of a subroutine with subsequent return is allowed. See section 8.)

(j) DO loops may be nested; this situation is considered in the next section on subscripted variables.

6.8 FOR statements

A more powerful statement for controlling loops is the FOR statement. This has the general form

FOR n X = A_1, A_2, A_3

and calls for a set of statements to be executed with the index X assuming a range of values: it differs from the DO statement in the following ways:-

(a) The last statement in the range may be indicated by a matching statement REPEAT instead of a numbered statement. In this case n is omitted. This procedure is recommended.

(b) The index may be an INTEGER variable and the indexing parameters INTEGER expressions, or the index may be a REAL variable and the parameters REAL expressions.

(c) The number of executions of the loop is determined as follows. For INTEGER index and parameters, if $A_2 - A_1$ is of opposite sign to A_3 then the only effect is to set the value of $X = A_1$ and the range is not executed at all, that is, it is skipped. If $A_2 - A_1$ is zero or has the same sign as A_3 then the range is executed with the index having the values A_1, $A_1 + A_3$, $A_1 + 2A_3$, ... successively, so long as $A_2 - X$ is zero or has the same sign as A_3. For REAL index and parameters, the procedure is similar, but execution continues until $|(A_2 - X)/A_3| < 0.5$. The essential point here is that slight round-off error in the binary representation of real numbers is accounted for, so that for instance the statement

FOR X = 1.0, 2.0, 0.2

ensures execution with X having in turn the values 1.0, 1.2, 1.4, 1.6, 1.8, 2.0, there being no possibility of execution with X = 2.0 being omitted.

As with the DO statement, if the increment A_3 is 1 or 1.0 it may be omitted from the FOR statement.

6.9 Further examples

(a) DO and FOR statements are probably most powerful when used in connection with arrays and subscripted variables. Example 7 of section 2, however, although expressed using subscript notation, does not involve the use of subscripted variables if the numbers x_i are read in one at a time using a read instruction inside a loop, as in the block diagram in Figure 7.

Possible sections of program are:

```
      SUM, SUMSQ = 0
      DO 1 I = 1, 80
      READ 2, X
    2 FORMAT (F 10.0)
      SUM = SUM + X
      SUMSQ = SUMSQ + X**2
    1 CONTINUE
      XBAR = SUM/80
      X = SQRTF (SUMSQ/80)
```

or

```
      SUM, SUMSQ = 0
      FOR I = 1, 80
      READ 2, X
    2 FORMAT (F 10.0)
      SUM = SUM + X
      SUMSQ = SUMSQ + X**2
      REPEAT
      XBAR = SUM/80
      X = SQRTF (SUMSQ/80)
```

Notice that, in contrast to the previous example, the loop index is used only to control the number of repetitions and is not used by any statement in the loop.

(b) As a second example, consider a program to produce the following table of values. A function f(x) is defined by

$$f(x) = \tfrac{1}{2}x\sqrt{(b^2 - x^2)} - \tfrac{1}{2}b^2 \log (x + \sqrt{(b^2 - x^2)})$$

It is required to read from one card a value of a parameter b and an integer n and to tabulate f(x) for a range of values of x from 0 to b in steps of b/n . After each tabulation the program is to loop back to read new values of b and n , and to terminate when a value b = 0 is encountered.

A possible program using a DO loop is as follows.

```
5 READ 1, B, N
1 FORMAT (F10.0, I3)
  IF (B), 2,
  BSQ = B ** 2
  FB = BSQ/2 * LOGF(B) π F(0) = F(B)
  X = 0
  PRINT 3, X, FB
  DO 4 I = 1, N - 1
  X = X + B/N
  ROOT = SQRTF (BSQ - X ** 2)
  FX = X/2 * ROOT - BSQ/2 * LOGF (X + ROOT)
  PRINT 3, X, FX
4 CONTINUE
  PRINT 3, B, FB
3 FORMAT (2E 20.8)
  GO TO 5
2 CALL EXIT
  END
```

It may be noted that in the (unlikely!) event that n = 1 the above program will give one too many printed lines of output, the DO loop being executed once unnecessarily. In the following program using a FOR statement, even this contingency is catered for.

```
5 READ 1, B, N
1 FORMAT (F10.0, I3)
  IF (B), EXIT,
  BSQ = B*** 2
  FB = BSQ/2 * LOGF (B)
  X = 0
  PRINT 3, X, FB
  FOR X = B/N,(N-1) * B/N, B/N
  ROOT = SQRTF (BSQ - X ** 2)
  FX = X/2 * ROOT - BSQ/2 * LOGF (X + ROOT)
  PRINT 3, X, FX
  REPEAT
  PRINT 3, B, FB
3 FORMAT (2 E 20.8)
  GO TO 5
  END
```

Notes (a) The use of subroutine names as successors described in paragraph 6.4 may be available in later versions of Orion FORTRAN.

(b) In the first versions of Orion FORTRAN only the DO statement is available for loop control, with the restriction that if the increment is negative, it must be a negative constant. In later versions this restriction will be removed, and the FOR statement provided.

Exercises 6

Some of the following exercises involve writing a few statements only, which would form part of some larger program. Others require complete programs which should be written as such with the necessary declarations and END statements.

1. Write statements to perform the following steps:
   (i) to transfer to statement number 5 if QNEW - QOLD $\geqslant 10^{-8}$; otherwise to proceed to statement number 50.
   (ii) to transfer to statement number 1 if J < 0 , to proceed to the next statement if J = 0 and to transfer to statement number 2 if J > 0 .
   (iii) to place the smallest (in absolute value) of Y and Z in LOW.
   (iv) to place the algebraically largest of W, X, Y, Z in TOP using (a) three IF statements (b) any desired standard function.
   (v) to transfer to statement 9 if A1 < A2 < A3; otherwise to transfer to statement 10.
   (vi) if I = 0, to transfer to statement 12; if I = 1, 3 or 5 to transfer to statement 13; if I = 2 or 4 to transfer to statement 14; if I = 6, 7 or 8 to transfer to statement 15; if I < 0 or I > 8 to transfer to statement 1000.

2. Write a program to read the three coefficients of a quadratic equation $ax^2 + bx + c = 0$ and print out the roots on one line. The program is to loop back to read another set of coefficients and print the roots on the next line continuing until a value $a = 0$ is encountered indicating the end of the data. If the roots are complex, two zeros are to be printed in place of the roots.

3. Write a program to tabulate the values of the integral

$$I = \int_0^{\frac{\pi}{2}} \sin^n x \, dx$$

using the formulae below, for values of n from 1 to 100 inclusive, printing the pairs n, I on successive lines.

$$I = 1 \text{ if } n = 1$$
$$= \frac{1}{2} \cdot \frac{3}{4} \cdot \frac{5}{6} \cdots \frac{n-1}{n} \cdot \frac{\pi}{2} \text{ if } n \text{ is even}$$
$$= \frac{2}{3} \cdot \frac{4}{5} \cdot \frac{6}{7} \cdots \frac{n-1}{n} \text{ if } n \text{ is odd.}$$

4. Write statements to sum the following series.

(i) $z - \frac{z^3}{3!} + \frac{z^5}{5!} -$ up to and including the first term to have absolute magnitude $< 10^{-8}$.

(ii) $\sum_{n=1}^{20} (-1)^{n-1} \frac{h^{2n-1}}{(2n-1)^2} \sin (2n-1)\theta$

5. Tabulate the function

$$T(x) = \tan x + \tfrac{1}{3} \tan^3 x$$

for values of x from 0 to 89^0 in steps of one degree, printing pairs of values x, T on successive lines.

7. ARRAYS. NESTS OF DO AND FOR STATEMENTS

7.1 Subscripted variables

In section 2, paragraph 2.5.5 the use of arrays and subscripts was introduced. The name of an array in FORTRAN may be any identifier chosen in the usual way. A particular element of an array is referred to by writing the name of the array, followed by a list of subscripts separated by commas and enclosed in parentheses. Such an element is termed a subscripted variable.

Examples

(a) X(20) refers to the 20th element of a one-dimensional array (a vector) called X.

(b) If a two-dimensional array B is considered as a matrix, the first subscript may refer to the rows of B and the second subscript to the columns of B. Thus B(3,7) is the element in the third row and seventh column.

Arrays may have any number of dimensions; in each dimension the subscripts are numbered from one upwards.

Subscripted variables may be used, with very few exceptions, wherever simple variable names may be written: in particular, in input/output lists and arithmetic expressions.

Examples

```
READ 10, A(1), A(2), A(3), TIME, BETA (4,3)
W = (TEMP(2)+TEMP(1))/2
```

The power of the subscript notation arises from the fact that in each subscript position may be written any INTEGER arithmetic expression (with no restrictions, so that the subscript expressions may themselves use subscripted variables); with the aid of DO or FOR statements, program loops may be written to operate on array elements in a systematic manner.

Example 1

Calculate the mean of 100 numbers $A_1, A_2, \ldots A_{100}$ stored as a one-dimensional array.

```
      SUM = 0
      DO 12 I = 1, 100
      SUM = SUM + A(I)
   12 CONTINUE
      MEAN = SUM/100
```

Example 2

Given a vector $X_1, X_2, \ldots X_{20}$, form the vector Y with 20 elements defined by:

$$\begin{cases} Y_1 = X_1 \\ Y_i = \dfrac{X_{i-1} + 2X_i + X_{i+1}}{4} & 2 \leqslant i \leqslant 19 \\ Y_{20} = X_{20} \end{cases}$$

```
Y(1) = X(1)
FOR I = 2, 19
Y(I) = (X(I-1) + 2*X(I) + X(I+1))/4
REPEAT
Y(20) = X(20)
```

Example 3

A square matrix of REAL elements called MASS has dimensions 10 by 10. Put the element of largest modulus on the principal diagonal in MAX. (The principal diagonal of a matrix consists of those elements having the same row and column numbers.)

```
MAX = 0
FOR I = 1, 10
IF(ABSF(MASS(I,I)) - ABSF(MAX)) 99,99,
MAX = MASS (I,I)
99 REPEAT
(next statement of program)
```

7.2 Modes and dimensions of arrays

All the elements of an array must have the same mode; for instance either all INTEGER or all REAL. As with simple variables, the mode is defined by the initial letter of the identifier unless a specific mode declaration is made.

Thus in example 3 above, if the elements of MASS were real variables the declaration

```
REAL MASS, MAX
```

would be required.

The dimensions of an array must be declared by a DIMENSION statement, so that the compiler can distinguish the name from that of a function and reserve storage space for all elements of the array. For the examples above the statements

```
DIMENSION A(100)
DIMENSION X(20), Y(20)
DIMENSION MASS (10, 10)
```

would be required. Note that several arrays may be dimensioned in one statement. Like mode declarations, dimension declarations will usually be placed at the head of a routine. The subscripts appended to array names in a DIMENSION statement give the maximum sizes of subscript which will be expected during execution. No subscript which is zero or negative, or exceeds the limit given in a DIMENSION statement, should occur during execution, and the number of subscripts attached to a subscripted variable should be the same as the number given in the DIMENSION statement.

It is permissible and preferable to include dimension information in a mode declaration when one is made. Thus in example 3 the single statement

```
REAL MASS (10, 10), MAX
```

will serve to declare both modes and dimensions. The presence of a dimension declaration in one or other form is the only indication to the compiler that an identifier refers to an array.

7.3 Nested loop statements

The range of a DO or FOR statement may contain another DO or FOR statement controlling an inner loop. Such loops are said to be *nested*.

Example 4

Form the vector S, length 6, whose elements are the means of the columns of the matrix T, size 8 by 6.

```
DO 3 J = 1, 6
SUM = 0
DO 4 I = 1,8
SUM = SUM + T(I,J)
4 CONTINUE
S(J) = SUM/8
3 CONTINUE
```

The same index cannot be used to control two loops when one is nested inside the other, but the index of one DO or FOR statement may appear in the indexing parameters of an inner DO or FOR statement.

Example 5

An array ALPHA, size 10 by 10, is to be modified by changing the sign of all elements $ALPHA_{i,j}$ for which $i > j$, and making $ALPHA_{i,i}$ zero.

```
FOR I = 1, 10
FOR J = 1, I-1
ALPHA (I,J) = -ALPHA(I,J)
REPEAT
ALPHA(I,I) = 0
REPEAT
```

(This example illustrates rule (c) of paragraph 6.8 concerning FOR statements: during the first execution of the outer loop with I = 1, the FOR statement for the inner loop is equivalent to FOR J = 1,0,1 which is executed zero times. Therefore control passes direct to ALPHA(I,I) = 0. At the next execution of the outer loop with I = 2, the inner loop is executed once with J = 1 to change the sign of ALPHA (2,1) and so on.)

There is no theoretical limit to the "depth" of nesting allowed. In practice, the only restrictions are due to limited working space in the compiler. The rules for single DO and FOR statements apply, with two additions. Firstly, the range of an inner statement must be completely contained within the range of an outer statement, and two FOR statements must have two distinct REPEAT statements. (It is recommended also that DO statements have distinct CONTINUE statements.)

Example 6

An array THETA has size 5 by 9. Form the array TAN whose elements are the tangents of the corresponding elements of THETA.

```
FOR I = 1,5
FOR J = 1,9
TAN(I,J) = TANF(THETA(I,J))
REPEAT
REPEAT
```

Secondly, the rule that a jump into the range of a DO or FOR statement from outside is not permitted does not preclude leaving an inner loop by an IF or GO TO statement leading to a statement in an outer loop. This is possible, of course, because from the standpoint of the outer loop, the transfer is entirely within its range.

Example 7

Taking $x = 0, 0.1, 0.2, \ldots, 1.0$ in turn, print out values of the expression

$$\sqrt{2 - e^{xy^2}}$$

for $y = 0, 0.2, 0.4, \ldots$ (For each value of x, there is a value of y which, if exceeded, makes $2 - e^{xy^2}$ negative; calculation may then cease for that value of x. An upper limit of $y = 1.4$ is sufficient even for $x = 0$)

```
   FOR X = 0, 1.0, 0.1
   E = EXPF(X)
   FOR Y = 0, 1.4, 0.2
   R = 2 - E*Y**2
   IF(R) 91 ,,
   ROOT = SQRTF(R)
   PRINT 10, X, Y, ROOT
   REPEAT
91 REPEAT
10 FORMAT (3F14.6)
   GO TO EXIT
   END
```

The block diagram (Figure 15) illustrates how control passes from the inner to the outer loop if R is negative before the inner loop has been executed the full number of times.

7.4 Input and output of arrays

The elements of an array may be input or output by naming each element individually, using subscripts. For example, the statement

```
READ 20, A(1), A(2), A(3)
```

will read the three elements A_1, A_2, A_3 of the array A according to FORMAT statement 20. Such a method is tedious in all but the simplest cases; it may be useful if a few elements of an array are required, perhaps in some odd order. For example,

```
READ 18, A(I), W(100), W(400)
```

By quoting an array name in an input/output list without subscripts the entire array may be called. The order in which the elements are dealt with is the order in which they are actually stored, which is such that the first subscript varies most rapidly, the second subscript next most rapidly and so on. For example, if Z is an array of size 5 by 3, the statement

```
READ 9, Z
```

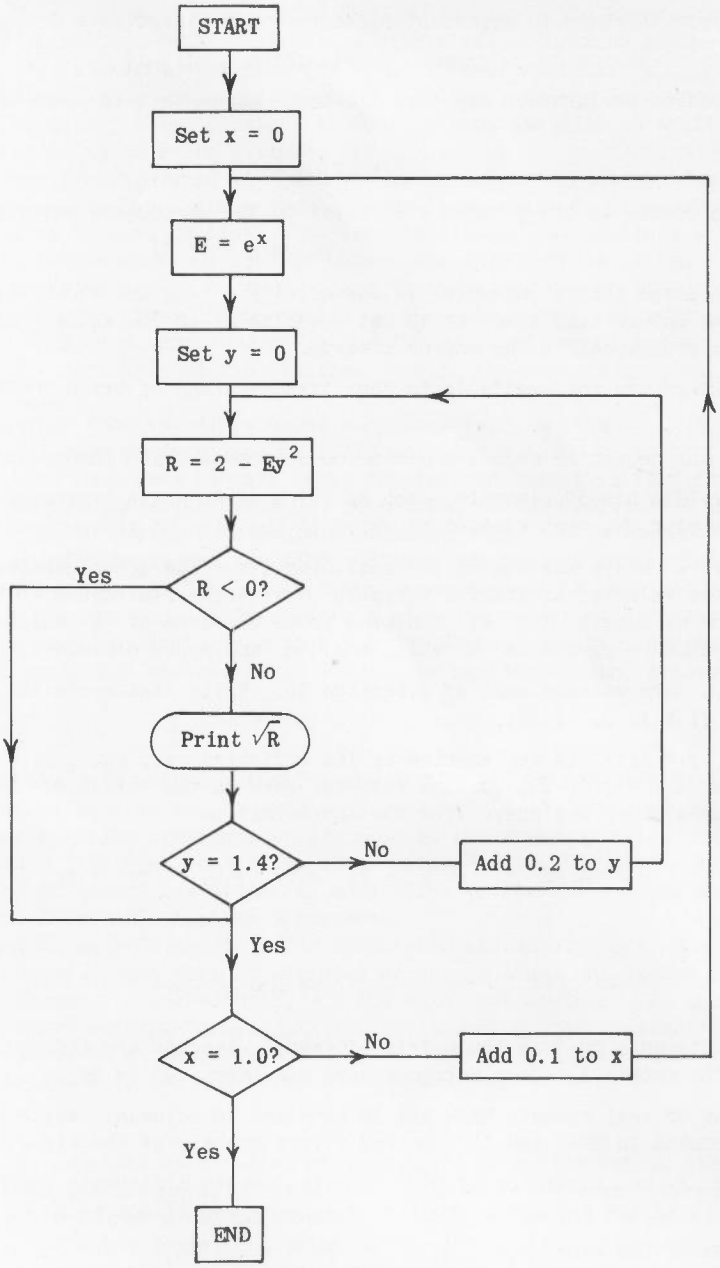


Fig. 15

will call for the elements in the order

$$Z_{1,1},\ Z_{2,1},\ \ldots\ Z_{5,1},\ Z_{1,2},\ Z_{2,2},\ \ldots\ldots\ \ldots\ldots,\ Z_{5,3}.$$

This is equivalent to obeying the following nested FOR statements:

```
FOR J = 1,3
FOR I = 1,5
READ 11, Z(I,J)
REPEAT
REPEAT
```

with one important difference. Since each execution of an input statement starts a new record, use of these nested statements would require each element to be contained in a separate record; for example, each element would have to be punched on a separate card for punched card input. Using the simple statement READ 9, Z allows more than one element to be in one record, if the FORMAT statement 9 is suitably specified, as discussed in section 10.

7.5 Parametric dimensions

In this section is has so far been assumed that the dimensions of arrays are stated in the source program as constants, and thus fixed at time of compilation. It is possible to declare dimensions using *parameters*, whose values are not fixed until each occasion when the compiled program is loaded prior to execution; this may make it simpler to write some programs.

Identifiers intended to represent parameters are listed in a declaration such as

```
PARAMETER I, J1, J2
```

An array dimension declaration may then use these parameters in place of constant dimensions; for example,

```
DIMENSION ARRAYB (J1, J2, I)
```

and other statements in the program will refer to the parameters where necessary, as for instance

```
DO 20 K = 1, J1
```

It must be stressed that a parameter is essentially a constant whose value is declared not in the source program but at load time; it is not a variable, in the sense that no assignment of value may be made to it by statements in the source program.

Note: PARAMETERS are not available in the first versions of Orion FORTRAN.

Exercises 7

Include any necessary mode and dimension statements, and choose suitable identifiers as necessary.

1. A vector BETA has 50 elements, each of which is an angle expressed in degrees. Write statements to form the vector SN, each element of which is the sine of the corresponding element of BETA.

2. A vector k has a maximum of 100 real elements. The actual number of elements during execution is given by the value of an integer variable N . Write statements to "split" k into two vectors k1 and k2 each of length N ; k1 contains those elements of k which are $\geqslant$d, and zeros elsewhere; k2 contains those elements of k which are <d and zeros elsewhere.

3. T and U are vectors each of dimension 30. Write statements to form the vector S = T − U i.e. $S_i = T_i - U_i$ (i = 1, 2, .. 30).

4. A curve y = f(x) is represented by its ordinates $y_1, y_2, \ldots\ldots y_{3N+1}$ at (3N+1) equally spaced points $x_1, x_2, .. x_{3N+1}$. The interval between the values of x is h . Write statements to compute the area under the curve from the approximation

$$\text{Area} = \frac{3h}{8}(y_1 + 3y_2 + 3y_3 + 2y_4 + 3y_5 + 3y_6 + 2y_7 + \ldots$$

$$\ldots + 3y_{3N-1} + 3y_{3N} + y_{3N+1})$$

$$(1 \leqslant N \leqslant 100)$$

5. Write statements to form the matrix ASQ whose elements are the squares of the corresponding elements of the matrix A. Both matrices have dimensions 40 by 50.

6. The array of real numbers MASS has 20 rows and 15 columns. Write statements to put the element of largest modulus in MMAX and the row and column numbers of the element in ROWNO and COLNO respectively.

7. A vector X has a maximum of 200 elements. Write statements to check whether the first N elements are in strict ascending order of magnitude and if not to go to statement number 99.

8. The value of the expression

$$\frac{1}{\sqrt{(1 - e^2)}} \cos^{-1} \frac{e + \cos x}{1 + e\cos x}$$

is to be evaluated for all combinations of e and x , e taking the values -0.9, -0.8, -0.7, ... 0.9 and x taking the values 0, 0.1, 0.2, ... 3.0. Write statements to store the values as an array with 31 rows corresponding to the values of x and 19 columns corresponding to the values of e .

9. A and B are matrices with dimensions I by J and J by K respectively, the maximum dimensions being 50 by 50. Write statements to form the matrix C, dimensions I by K , obtained by multiplying A and B . Matrix multiplication may be defined by

$$C_{rs} = \sum_{p=1}^{J} A_{rp} B_{ps} \qquad \begin{matrix} r = 1, 2, \ldots I \\ s = 1, 2, \ldots K \end{matrix}$$

8. FUNCTIONS AND SUBROUTINES

8.1 Program structure

In the previous sections emphasis has been placed on the structure of single routines. The only functions considered have been those provided by the system (section 3). Whenever such a function is called for by its appearance in an expression, followed by an argument or arguments, the compiler includes in the object program the necessary links to a set of machine instructions for evaluating the function, without further specification from the programmer. FORTRAN has facilities which enable a

programmer to name and define his own functions; such functions must be similar to the system functions in so far as they operate on one or more arguments to provide a single value for use in expressions. More generally, the programmer may write sections of program in the form of subroutines to perform computations more powerful than the evaluation of a single function value, and may link up these subroutines in various ways. Consequently, only the simplest FORTRAN programs will be written as single routines, the usual procedure being to write a linked set.

There are three types of routine distinguished in FORTRAN, namely the main routine, functions and subroutines. Routines are *separately* compiled, into a form of machine language together with symbolic information required for communication between routines. At execution time, all routines are loaded by a special program which sets up the intercommunications and enters the main routine at its first executable statement. The main routine is always unnamed; other routines are named, by identifiers chosen in the usual way, and may call each other and be called by the main routine by name in an arbitrarily complicated manner, so long as a routine does not call itself, either directly or indirectly via another routine call. Obviously no two routines used in the same program may have the same name.

Several important advantages arise from the division of a program into routines. If a change is required to a program for any reason, only the routines directly affected need to be re-compiled. Short routines are acceptable and may indeed lead to a shorter total compilation time than long routines; programmers are therefore encouraged to make alterations and corrections to the original FORTRAN statements and not to tamper with the compiled routines in machine language. The routine becomes a natural unit of programming and the division of a large program into routines can reflect the logical structure of the flow chart; routines may be developed, written and tested independently of each other, possibly by different programmers. The main routine may itself perform a sizeable amount of computation, or may be merely a skeleton or controlling program linking together other routines, at the programmer's convenience. It is suggested that up to 100 statements is a suitable routine length, although no upper limit is set.

8.2 Properties of routines

The names of variables and labels used within a routine are normally *local* or *private* to that routine; that is, they have no meaning outside the routine (and so may be used for completely different purposes in another routine) unless specific arrangements are made to the contrary using the PUBLIC declaration (section 8.6.1). Certain fairly obvious rules apply within a single routine: for instance, the same identifier may not be used to denote two different quantities, statement numbers must be all distinct and the mode of a particular variable applies throughout.

When any routine is called, it is entered at its first executable statement; mode, dimension and other declarations will normally appear before this at the head of the routine. All paths of control should terminate at statements which cause a proper exit. In the main routine these are usually CALL EXIT or GO TO EXIT, or possibly control statements using EXIT as a successor name. After a function or subroutine has been executed control is normally returned to the calling routine. This is the effect of the statement RETURN introduced in the following paragraphs.

8.3 Function routines

A routine written as a FUNCTION consists of a set of statements defining the way in which the value of a function is to be computed. The definition will frequently be in terms of dummy (formal) arguments, for which actual arguments are substituted when the function is invoked. This is the approach adopted when using the standard functions, for which definition is automatically supplied; for instance, the function SINF(X) will compute the value of the sine of any arithmetic expression substituted as an actual argument in place of the formal argument X. Thus the appearance of SINF(2*PI*OMEGA*T) in an expression in one routine requires the value of 2*PI*OMEGA*T to be computed and transmitted to the SINF routine, which will evaluate the sine and return the necessary value to the calling routine.

The form of a function is illustrated by the following example.

Example 1

It is required to evaluate, at several points in a program, the area of a plane triangle from the lengths of its three sides. Calling the sides A, B, C the formula is

$$\text{AREA} = \sqrt{(S(S-A)(S-B)(S-C))}$$

$$\text{where} \quad 2S = A + B + C.$$

The following routine defines the function to compute the area

```
FUNCTION AREA (A, B, C)
S = (A+B+C)/2
AREA = SQRTF(S*(S-A)*(S-B)*(S-C))
RETURN
END.
```

The following features should be noted. A function routine is always introduced by a statement of the form

```
FUNCTION NAME (A1, A2, A3, ...)
```

where NAME is the particular name of the function and A1, A2, A3, ... are the dummy arguments. In the example there are three dummy arguments A, B, C; there must always be at least one. After the FUNCTION statement are written any necessary declarations of mode, dimensions and so on, followed by the statements actually defining the computation. At some point in the routine a value must be given to the scalar variable having the same name as the function. In the example, the third statement assigns a value to AREA. (The mode of the result of the function is determined by the first letter of the function name in the usual way, unless a mode declaration to the contrary appears. If the above function were named LAMDA, for example, the routine would read

```
FUNCTION LAMDA (A,B,C)
REAL LAMDA
S = (A + B + C)/2
LAMDA = SQRTF (S*(S-A)*(S-B)*(S-C))
RETURN
END.
```

and in any routine calling this function the declaration

```
REAL LAMDA
```

would also be required.) The statement RETURN causes transfer of control back to the point at which the calling routine is left; in the case of a function routine this is back into the evaluation of some expression. The END statement as always indicates to the compiler the physical end of the routine. The RETURN statement need not be immediately before the END statement and may be labelled; there may be more than one RETURN statement.

An example of use of the above function is

```
P = AREA (2*B, R(2), 10.0) + AREA (S, B, R(1))
```

This causes the sum of the areas of two triangles, whose sides are of lengths 2B, R_2, 10.0 and S, B, R_1 respectively, to be assigned to P , the function AREA being called twice. The variables B and S appearing in this statement are variables in the calling routine, and have no connection with B in the function routine, which is simply a dummy argument, nor with S which is a local working variable.

8.4 Subroutines

A SUBROUTINE is a more general purpose routine which differs from a FUNCTION in three ways. Firstly, a subroutine is not restricted to producing a single value, or to having at least one argument. It may be a section of program to perform any desired computation: it may, for instance, set its results in arrays. Secondly, whereas a function is invoked by its appearance in an expression in the calling routine, a subroutine is invoked by a specific statement of the form

```
CALL NAME (A1, A2, A3, ..)
```

where NAME is the name of the subroutine and A1, A2, A3, .. are actual arguments. If a subroutine has no arguments then the list (and the parentheses) are omitted. Thus EXIT is a library routine which expects no arguments and may be called by the statement CALL EXIT. As previously noted, in Atlas FORTRAN if a routine has no arguments and no return to the calling routine is required, a statement such as GO TO EXIT may equally well be used. Thirdly, the statements of a subroutine are introduced by a statement having the general form

```
SUBROUTINE NAME (B1, B2, B3, ...)
```

where B1, B2, B3, ... are the dummy arguments, if any.

Example 2

The following is a subroutine to compute the formulae of example 14 of section 2.

```
      SUBROUTINE PQ (P, Q, Z)
      IF (Z - 2000)1,,
      P = 1
      GO TO 2
    1 P = 1 + SQRTF (4 - .002*Z)
    2 IF (Z - 1000)3,,
      Q = .0035*Z + 0.5
      RETURN
    3 Q = .0045*Z - 5E-7*Z**2
      RETURN
      END
```

This subroutine has one input argument Z ; the arguments P and Q are referred to as output arguments. When the subroutine is called, actual variable names will be supplied corresponding to P and Q to indicate where the computed results are to be stored. For example, a possible call is

```
CALL PQ (T, P, H + N*DELTA)
```

which requires the value of H + N*DELTA to be transmitted to the subroutine; the two results will be assigned to the variables T and P in the calling routine.

A point to be noted concerns dummy arguments which appear many times within a function or subroutine. When this situation arises the compiled routine will generally be faster if a single assignment of the value is made to a local variable of the routine, and this local variable used wherever necessary in subsequent statements. The above routine, for instance, may be better written as follows:

```
SUBROUTINE PQ (P, Q, ZZ)
Z = ZZ
IF (Z - 2000) 1,,
      etc.
```

8.5 Correspondence between dummy and actual arguments

There must be a close correspondence between the dummy arguments used in a routine definition and the actual arguments supplied by a calling routine, summarised in the following paragraphs. Recalling that individual routines are compiled independently, it is evident that violation of the rules cannot be checked by the compiler, and may cause chaos when the program is executed.

Actual arguments and dummy arguments should be the same in number, presented in the same order and be of the same type and mode.

An actual input argument, corresponding to a dummy argument which is a simple variable, may be a constant, a variable - simple or subscripted - or any general expression of the same mode, but clearly if the actual argument is an output argument it may be only a variable - simple or subscripted.

Arrays may be used as arguments. The dimensions of actual and dummy arrays must be in agreement; if they are constant dimensions then they must be the same in both the called and the calling routine.

Example 3

```
SUBROUTINE TRANSP (A)
DIMENSION A (100, 100)
FOR I = 1, 99
FOR J = I + 1, 100
W = A (I,J)
A (I,J) = A (J,I)
A (J,I) = W
REPEAT
REPEAT
RETURN
END.
```

This might be called by a routine containing the statements

```
DIMENSION BETA (100, 100)
CALL TRANSP (BETA)
```

which would have the effect of transposing the array BETA.

It is possible for the dimensions of *dummy* arrays to be *adjustable*; at each call they must be adjusted to agree with the actual arguments. Using adjustable dimensions, the above subroutine may be written thus:

```
SUBROUTINE TRANSP (A,N)
DIMENSION A (N,N)
FOR I = 1, N - 1
FOR J = I + 1, N
W = A (I,J)
A (I,J) = A (J,I)
A (J,I) = W
REPEAT
REPEAT
RETURN
END.
```

This might be called by a routine containing the statements

```
DIMENSION BETA (60,60)
CALL TRANSP (BETA, 60)
```

It may be noted here that because no storage needs to be assigned for a dummy array, a declaration of dimensions in a subroutine or function, apart from indicating that an identifier is the name of a dummy array, is required by the compiler only for the purpose of assembling instructions to locate the position of an element of an array in storage relative to the first element. A little thought should convince the reader that, since array elements are stored in the order obtained by varying the first subscript fastest, the last (or only) dimension of a dummy array may be any number, conveniently 1.

Example 4

```
FUNCTION XBAR (X,N)
DIMENSION X(1)
SUM = 0
FOR I = 1,N
SUM = X(I) + SUM
REPEAT
XBAR = SUM/N
RETURN
END
```

A dummy argument may be a dummy subroutine name or a dummy function name, the corresponding actual argument being a subroutine name, or a function which is not the name of one of the "open" standard functions. The reader is referred to the manuals for further details.

8.6 Communication between routines

Besides the use of arguments, two other methods exist in FORTRAN for communicating values from one routine to another; their purpose is to make the location of certain variables directly available to more than one routine.

8.6.1 PUBLIC variables. As previously stated, the names of variables are normally private to individual routines; however, by declaring certain variables to be PUBLIC their names can be made available outside the scope of a single routine.

Thus the declaration

```
PUBLIC X,Y
```

will enable any routine in which it appears to refer to the same variables X and Y . Dimensions may be included in a PUBLIC declaration; for instance

```
PUBLIC X, Y, Z (100), ARRAYB (5, 5, 10)
```

There is no significance in the order of appearance of variables in a PUBLIC declaration.

It should be realised that a routine not declaring, say, Z to be PUBLIC may use the identifier Z for any private purpose without causing confusion of names.

A PUBLIC array may have parametric dimensions; a dummy array may not of course be declared PUBLIC.

8.6.2 COMMON storage. A certain area of storage is considered COMMON to all routines of a program. Variables stored in this area will be available to all routines because they occupy a known storage position, not because of any identity of names.

For instance, if one routine contains the declaration

```
COMMON X, Y(100), Z(10), ZA, YB(2,5,3)
```

then these variables will be stored in the COMMON area starting from a fixed location, in the order

```
X, Y(1), Y(2), ..., Y(100), ..., Z(10), ZA,
YB(1,1,1), YB(2,1,1), ..., YB(2,5,3)
```

If in another routine the declaration

```
COMMON AW(3), AZ(20), AY(40)
```

appears, then the same storage is occupied by the variables

```
AW(1), AW(2), AW(3), AZ(1), ... AZ(20), AY(1), ..., AY(40)
```

in that order. Thus X and AW(1) are references to the same variable, as are Y(1) and AW(2) and so on. Consequently, a value of X left by the first routine may be referred to by the second routine by the name AW(1) and so on. As will be appreciated, changes to COMMON storage between routines must be carefully handled, or chaos may result. In practice variables, particularly arrays, required in several routines are usually assigned to COMMON in the same order in each routine, so that the COMMON statements are identical in all routines; it then appears that variables are being communicated by name, although this is not in fact so.

Arrays assigned to COMMON storage must have constant dimensions; a dummy array is never declared COMMON.

Example 5

```
SUBROUTINE MATMPY
COMMON DUMMY (25), I,J,K,U(30,30), V(30,30), W(30,30)
FOR N1 = 1,I
FOR N2 = 1,J
W(N1,N2) = 0
FOR N3 = 1,K
W(N1,N2) = U(N1,N3)*V(N3,N2) + W(N1,N2)
REPEAT
REPEAT
REPEAT
RETURN
END
```

to be used within a routine containing the statements

```
COMMON A,B,IN1,IN2,IN3,MATA,MATB,MATC
DIMENSION A(20), B(5), MATA(30,30), MATB(30,30), MATC(30,30)
IN1 = 15
IN2 = 25
IN3 = 15
CALL MATMPY
```

The arrays A and B together occupy 25 storage locations, accounted for in the subroutine by the dummy array DUMMY. The scalars IN1, IN2, IN3 are in the same storage as I, J, K of the subroutine, and so on.

8.7 EQUIVALENCE

The placing of variables in COMMON storage not only provides one means of communication of values between routines, but also enables data storage area to be shared between routines. When the logical structure of a program allows, this can result in a large saving of storage space.

An analogous method allowing storage to be shared within a routine is provided by the EQUIVALENCE statement. For instance, the statement

```
EQUIVALENCE (A,B,C), (D,E,F,G)
```

causes A, B and C to share one storage location and D, E, F and G to share another. Both simple variables and arrays may be equivalenced; any number of variables may appear within one pair of parentheses and there may be any number of parentheses. Private variables not mentioned in an EQUIVALENCE statement will be assigned separate, unique storage locations. Dummy variables may not appear in an EQUIVALENCE statement. At most one PUBLIC variable or variable assigned to COMMON storage may appear in each pair of parentheses.

The reference manuals give full details of the possible use of this statement. An incidental advantage is that it allows one variable to be referred to by more than one name, which is occasionally a convenience.

8.8 Local functions

If a function, which can be defined by a single FORTRAN statement, is required only in one particular routine, it may if desired be defined at the head of that routine rather than as a separate function routine.

Examples

```
YOURF(X) = LOGF(1 + SQRTF(1 - X**2))

FN1 (X,Y) = (X**2 + Y**2)**(4 + I)
```

The name of the function determines the mode of the function in the usual way. The right-hand side defining the function in terms of dummy arguments may be any arithmetic expression not involving subscripted variables, but involving perhaps standard functions, function routines or local functions previously defined. Any variables appearing on the right-hand side which are not dummy arguments are assumed to be variables appearing in the routine itself.

When the function is called, these variables take their current values and actual arguments are supplied in place of dummy arguments. The use of local functions is thus very similar to the use of standard functions. Possible calls of the above functions are, for instance

```
A = B + YOURF(B - 1)

C = SINF(FN1(2*D,E))
```

Note: PUBLIC variables and local functions are not available in the first version of Orion FORTRAN.

Exercises 8

1. (i) Write a function routine to define the function

$$\tanh(x) = \frac{e^{x} - e^{-x}}{e^{x} + e^{-x}}$$

assuming the library function TANHF is not available.

(ii) Write a FORTRAN statement using this function to evaluate

$$a = b^2 + 2\tanh\left\{\frac{b}{\sqrt{(b^2 + c^2)}}\right\}$$

2. (i) Write a function routine to define the function

$$\text{sech}(x) = \frac{2}{e^{x} + e^{-x}}$$

(ii) The maximum shearing stress T_{max} in a twisted bar, whose cross-section is a rectangle of dimensions a and b, is given by

$$T_{max} = k \times 2G\Theta a$$

where G is the modulus of rigidity and Θ the twist per unit length. k is defined by the series

$$k = 1 - \frac{8}{\pi^2}\sum_{n=1,3,5,..}^{\infty}\frac{1}{n^2\cosh(n\pi b/2a)}$$

Write a section of program to calculate k to an accuracy of 6 decimal places, for values of the ratio b/a from 0.05 to 1.00 in steps of 0.05 storing the results as a one-dimensional array k with 20 elements. Use the function sech defined in (i).

3. Write a function routine to evaluate the function

$$k(x) = (\sin x - x\cos x)/x^3$$

accurate to at least 8 significant figures for all x. Hint: if x is small then sin x and x cos x are approximately equal and excessive cancellation will lead to a serious loss of accuracy when the subtraction is performed.

4. (i) Write a local function definition to compute the expression

$$e^{y} - \sqrt{1 - y^2}$$

(ii) Use this function in statements to compute

$$A = e^{2t} - \sqrt{1 - 4t^2} + \sin pt$$

and

$$B = \tan(\sqrt{1 - n^2} - e^{-n})$$

5. Write a subroutine to normalise a vector $x_i (i = 1,2,..n)$; that is, to find the element x_m of the vector having the largest absolute value and to divide every element of the vector by x_m. The length n of the vector is to be one of the arguments of the subroutine.

6. A program consists of several routines operating upon symmetric matrices with dimensions up to 100 by 100. Write one routine to determine the largest off-diagonal element of a matrix; this element and its subscripts are to be output arguments of the routine. The actual dimension (n by n) is to be an input argument.

7. Write a subroutine to check a square array for symmetry; that is, given $[A_{i,j}] (i = 1,2,..n; j = 1,2,..n)$ to call another subroutine ERROR if any pair of elements $A_{i,j}$ and $A_{j,i}$ $(i \neq j)$ are not equal to within 10^{-8} in absolute magnitude. The array is to have adjustable dimensions and the ERROR routine is to be a formal argument of the subroutine.

8. A program is to consist of five routines – a main routine linking four subroutines named R1, R2, R3, R4. The following variables are operated upon or calculated by more than one routine: three simple variables M, X, Y and four arrays A, B, C, D of dimensions (100), (10,10), (100,50) and (10,50) respectively. R1 uses M, X, Y and A; R2 uses M, X, A and C; R3 uses X, C and D and R4 uses M, B and D.

Assuming that none of these variables are communicated as arguments, indicate the declarations required in each routine (i) to place and use these variables in COMMON storage or alternatively (ii) to refer to the variables by PUBLIC names.

9. LOGICAL VARIABLES AND TEXT

The facilities introduced in previous sections are sufficient for writing many FORTRAN programs. However, in this section the means for performing simple logical operations on appropriate elements, and for handling TEXT, are outlined; in section 10 input/output facilities are considered in greater detail.

9.1 Logical values

A logical value is stored in both Atlas and Orion FORTRAN as a string of independent bits, each bit being either 1 or 0 representing a truth value true or false respectively; suitably declared variables may assume logical values and logical operators are provided to manipulate these bit-strings according to certain rules.

By convention a string of 1-bits is said to represent the value .TRUE. and a string of 0-bits to represent the value .FALSE. This primer is not concerned with operations on other bit-strings, but details may be found in the reference manuals.

9.2 Constituents of logical expressions

There is a close similarity between arithmetic expressions constructed from arithmetic constituents and operators, and logical expressions constructed from logical constituents and operators. Logical constituents may be logical variables, arrays and functions, logical constants and relations.

9.2.1 Logical variables, arrays and functions. These are named by identifiers chosen in the usual manner: a LOGICAL mode declaration is required. For instance, the declaration

```
LOGICAL MARK, Z (4)
```

declares the simple variable MARK and the elements of the array Z (dimension 4) to be LOGICAL quantities.

9.2.2 Logical constants. The symbols .TRUE. and .FALSE. are available as two standard logical constants.

9.2.3 Relations. Simple relations were introduced in section 6. The most general form of relation is

```
(A Ø B Ø C .....)
```

where A, B, C, ... are arithmetic expressions and Ø is one of the six relational operators .GT. .GE. .EQ. .LE. .LT. .NE. Note that a relation is enclosed in parentheses.

Examples:

(A .LT. B) This relation has the value .TRUE. if A < B and the value .FALSE. otherwise.

(W .GT. X .GT. Y .GT. Z) This relation has the value .TRUE. if W, X, Y, Z are in strictly decreasing order of magnitude and the value .FALSE. otherwise.

9.3 Logical operators

Four logical operators are provided, denoted by .NOT. .AND. .ER. (alternatively .NONEQ.) and .OR. The third operator is a contraction for 'exclusive or' or 'nonequivalence'.

These operators have the following effect. .NOT. B is .FALSE. if B is .TRUE. and .TRUE. if B is .FALSE.

A .AND. B is .TRUE. if both A and B are .TRUE. and .FALSE. otherwise

A .ER. B is .TRUE. if A and B have opposite truth values and .FALSE. otherwise

A .OR. B is .TRUE. if at least one of A and B is .TRUE. and .FALSE. otherwise.

The operations are summarised in the following table.

	A	.FALSE.	.FALSE.	.TRUE.	.TRUE.
	B	.FALSE.	.TRUE.	.FALSE.	.TRUE.
.NOT. B		.TRUE.	.FALSE.	.TRUE.	.FALSE.
A .AND. B		.FALSE.	.FALSE.	.FALSE.	.TRUE.
A .ER. B		.FALSE.	.TRUE.	.TRUE.	.FALSE.
A .OR. B		.FALSE.	.TRUE.	.TRUE.	.TRUE.

Within a logical expression, the hierarchy of operations is understood to be

```
.NOT.
.AND.
.ER.
.OR.
```

unless overridden by parentheses. Two operators must not appear in sequence: thus the parentheses are required in the expression A .AND. (.NOT. B)

9.4 Assignment of logical values

The value of a logical expression is assigned to a logical variable by a logical replacement statement analogous to an arithmetic replacement statement.

Examples:

```
MARK = (A .LT. B)
```

MARK assumes the value .TRUE. if A < B and the value .FALSE. otherwise.

```
Z(1) = (X.GT.Y) .AND. (Y.LE.Z)
```

Z (1) becomes .TRUE. only if both relations are .TRUE. when this statement is executed.

9.5 The logical IF statement

It was stated in section 6 that the logical IF statement may be used to test any logical expression L and has the general form

IF (L) n_1, n_2

Control is transferred to statement n_1 if L is .TRUE. and to statement n_2 if L is .FALSE. If L is a relation it need not be enclosed in extra parentheses.

Examples:

```
IF (A .LT. B) 5,6
IF (W .GT. X .GT. Y .GT. Z) 10, 20
IF (MARK) 1,
IF ((I .EQ. 0) .OR. (J .EQ. 0)) 7,12
```

9.6 An example of a logical function

The following routine defines a logical function QUERY, which has the value .TRUE. if the first N elements of a real vector are less than unity in absolute value, and .FALSE. otherwise.

```
      FUNCTION QUERY (A, N)
      LOGICAL QUERY
      DIMENSION A (N)
      QUERY = .TRUE.
      FOR I = 1, N
      IF (ABSF (A (I)) . LT. 1) 10,
      QUERY = .FALSE.
      RETURN
   10 REPEAT
      RETURN
      END
```

This function might be called in another routine by the statement

IF (QUERY (BETA, 15)) 7, 77

to test the first 15 elements of a vector BETA. This routine must contain the declaration LOGICAL QUERY.

9.7 TEXT

Sequences of characters, known as TEXT, are stored in the computer using 6-bit binary code to represent each character, 8 characters per word. In single word lengths TEXT may be manipulated using logical-type operations. Most commonly, however, TEXT is simply input, or included in the program, and output with no intermediate manipulation.

Variables and arrays may be named for holding TEXT by an appropriate mode declaration; for instance

TEXT TITLE (4), WORD

declares that the array named TITLE (length 4 words) and the simple variable WORD will be used to hold TEXT. The array will accommodate not more than 32 characters and WORD not more than 8 characters.

9.7.1 Constant TEXT. This may be included in a program in several ways. A TEXT constant can be written as nH , followed by n characters, where n is a positive integer. Any required blank spaces must be included in the count of characters. Such a constant may be specified for output by including it directly in a FORMAT statement - see paragraph 10.5.3 of the next section - or assigned to a TEXT variable e.g.

```
WORD      = 8HALPHABET
TITLE (1) = 5HGAMMA
```

In the second example the characters GAMMA will be left adjusted and the word TITLE (1) filled out with three blanks.

An alternative method of writing TEXT constants which is available and to be preferred in Atlas FORTRAN only, uses a special FORMAT statement. The constant is referred to by writing simply n A , where n is the number of a FORMAT statement having the form

n FORMAT π ..... π

All the characters, including blanks, between the two characters π constitute the constant. For instance,

```
      WORD = 98A
   98 FORMAT πALPHABETπ
      TITLE(1) = 99A
   99 FORMAT πGAMMAπ
```

9.7.2 Input and output of TEXT. Values of TEXT variables are input and output using an A-type field specification.

Example:

```
      TEXT TITLE (4)
      READ 10,TITLE
   10 FORMAT (4A8)
```

These statements will call for 4 groups of 8 consecutive characters each from the beginning of a record to be read and stored in the locations TITLE (1), TITLE (2), TITLE (3) and TITLE (4). Further examples are to be found in section 10.

Exercises 9

1. Write logical IF statements to have the following effects.
   (i) To transfer to statement 9 if $A1 < A2 < A3$ and $I \neq 1$ and to statement 10 otherwise.
   (ii) To transfer to statement 99 if Y lies outside the range $-3 \leqslant Y < 5$ and to proceed to the next statement otherwise.

2. The first record of an output document is to be one of the four phrases CASE A, CASE B, CASE C and NORESULT; the choice is determined by the value of an integer variable K computed during execution (K = 1, 2, 3 or 4).
   Write statements to set up the phrases as items in a TEXT array, and PUNCH and FORMAT statements to output the correct phrase.

10. FURTHER INPUT AND OUTPUT

In section 5 simple methods of using input/output statements in conjunction with FORMAT statements were introduced. This section indicates how certain straightforward extensions of the concepts of the input/output list and field specifications allow much more powerful methods to be used.

10.1 Choice of input/output medium

The most general forms of input/output statements are

```
READ  (a,f) list
WRITE (b,f) list
```

These specify not only the number (f) of the associated FORMAT statement, but also the logical number (a) of the input document or the logical number (b) of the output document. The choice of suitable numbers allows the use of magnetic tape and paper tape devices and other peripherals. If f is omitted then input or output using binary tapes is intended. Further details must be sought from the reference manuals: this primer is concerned only with the use of system documents by means of the special forms of statement

```
READ f, list
PUNCH f, list
PRINT f, list
```

introduced in section 5.

10.2 Input/output lists

A list consists of a series of items separated by commas. So far, these items have been simple items - that is simple variables, individual subscripted variables or array names without subscripts.

Compound items are also permissible, of the form

(list, $i = m_1, m_2, m_3$)

or (list, $i = m_1, m_2$)

where unity is implied for m_3. A compound item is a shorthand notation, analogous to the DO statement, calling for the list to be repeated with the index i taking in turn the values in the sequence $m_1, m_1 + m_3, m_1 + 2m_3, \ldots$; the rules are similar to those for DO statements.

Example 1

```
(A(I),I = 1,10)
```

This item is equivalent to the list

```
A(1), A(2), A(3),....,A(10)
```

Example 2

```
(I,A(I),B(I),I = 1,11,2)
```

This item is equivalent to the list

```
1,A(1),B(1),3,A(3),B(3),.....,11,A(11),B(11)
```

Since a compound item may be a member of a list, and a compound item calls for the repetition of a list, it is possible to nest compound items to virtually unlimited depth. Statements may thus call for the input or output of a large number of values: it is the function of the field specifications within the associated FORMAT statement to control the arrangement of these values on the external medium.

Example 3

```
((A(I,J), J=1,N),I=1,M)
```

This item is equivalent to the list

```
A(1,1),A(1,2),A(1,3),....,A(1,N),
A(2,1),A(2,2),.....  ,A(2,N),
.....
A(M,1),.................,A(M,N)
```

Example 4

```
M,(I,(W(I,J),J=1,5),X(I),I=1,M)
```

This list is equivalent to the list

```
M,
1,W(1,1),W(1,2),.....,W(1,5),X(1),
2,W(2,1),W(2,2),.....,W(2,5),X(2),
.....
M,W(M,1),W(M,2),.....,W(M,5),X(M)
```

Notes: (a) In section 7.4 it was stated that when an array name appears in a list without subscripts i.e. as a simple item, the whole array is being named for input or output in "natural order". This order is in fact equivalent to that specified by the compound item:

$$....(((ARRAY(I,J,K,.....),\ I = 1,d_1),\ J = 1,d_2),\ K = 1,d_3)....$$

where d_1, d_2, d_3 are the relevant dimensions.

(b) For input statements, elements to the left of bracketed elements in the list assume their new values before the bracketed elements are dealt with.

Thus K,(A(I),I=1,K) reads one number into K and then, using this value of K, reads A(1) to A(K).

To read values of I and J, and then an element A(I,J) requires an apparently superfluous pair of brackets thus:

```
I,J,(A(I,J))
```

10.3 FORMAT statements

An input/output list is scanned in conjunction with a list of field specifications given in the associated FORMAT statement in order to determine the mode of conversion of each item to or from its internal binary representation; such conversion specifications are termed class 1 field specifications. The FORMAT statement may also contain specifications, known as class 2 field specifications, which do not relate directly to items in a list: when these are encountered during the scan of the FORMAT statement they are acted upon without reference to the list, to control such features as constant text, horizontal or vertical spacing, zero suppression and scaling.†

10.4 Class 1 field specifications

Some of these have been introduced at appropriate points in previous sections, and are summarised in the following table: w denotes a field width, and d the number of decimal places where applicable.

Specifications	Mode of item in list	External form
Ew.d	REAL	Floating decimal
Fw.d	REAL	Fixed point decimal
Iw	INTEGER	Decimal integer
Gw (Atlas)	REAL	Dependent on number
Aw	TEXT	Text

Other specifications are available for the input and output of octal integers and logical quantities: these are not dealt with in this primer.

† Class 1 and class 2 field specifications are alternatively referred to as conversion specifications and controls respectively.

10.5 Class 2 field specifications

10.5.1 Repetition of class 1 specifications. Any of the class 1 specifications may be preceded by a positive integer n , to signify n repetitions of that specification. This feature was introduced in section 5. Furthermore, a group of field specifications may be enclosed in parentheses and preceded by an integer to signify repetition of the group. If n is not given it is assumed to be unity.

10.5.2 Scale factors. An E- or F-type specification may be preceded by sP, where s is a positive, negative or unsigned integer.

The effect on an E-type specification is to cause the mantissa to be brought into the range

$$10^{S-1} \leqslant |\text{mantissa}| < 10^{S}$$

and the exponent adjusted accordingly.

The effect on an F-type specification is defined by

$$\text{External value} = \text{Internal value} \times 10^{S}$$

In other words, there is an implied decimal exponent of -s.

Note that if no scale factor is given, the presumed settings are as follows:

On Atlas : 1P for E-type, 0P for F-type.

On Orion : 0P for E-type, 0P for F-type.

Example 5

Typical output using the specification 3PE20.5:

Value	Printed output
2.4075	240.75000E-02
-473.567812	-473.56781E-00
-0.1	-100.00000E-03

A scale factor affects all E- and F-type specifications which follow it; if more than one scale factor is given, each applies until the next one is reached. The ordering referred to here is the order of scan of the FORMAT statement (see paragraph 10.6). The presumed settings are restored when a new input/output statement is commenced.

10.5.3 Constant text. The specification nH , followed by n characters (possibly including blanks) calls for the output of these n characters exactly as written. A comma is not essential to separate these from the next specification. If this is the only specification in a FORMAT statement, no list is attached to the output statement.

Example 6

```
      PRINT 14
   14 FORMAT (24H THIS IS A PROGRAM TITLE)
```

Execution of the print statement causes the printing of the string of characters

```
|THIS IS A PROGRAM TITLE
```

on a new line. The blank first character is not printed - see paragraph 10.5.6.

Example 7

```
      PRINT 15, A
   15 FORMAT (7H ALPHA=F10.4)
```

A typical output would be, when A has the value 59.32567

```
|ALPHA=    59.3257
```

10.5.4 Blank fields. The specification nX specifies on output that n blanks are required, and on input that the next n characters are to be ignored. This need not be followed by a comma.

10.5.5 Separation of records. It has been stated previously that each input or output statement starts a new record e.g. a new card, or a new line of printed matter. The character / used in a FORMAT statement specifies the end of a record, and that what follows is to start a new record; (n+1) consecutive repetitions of the character / will produce n blank records.

10.5.6 Carriage control. For printed output, control of layout on the printed page is available using the first character of each record; provided this is one of a number of special control characters, it is not printed, but controls the vertical spacing as follows:

blank	single spacing before printing
0	double spacing before printing
+	no spacing i.e. two records printed on same line - available on Atlas only
1	record to be printed as the first on a new page.

The required character is frequently set using the nH specification; other characters are available in Atlas FORTRAN but are not considered here.

Example 8

```
      PRINT 100, SUMSQ
  100 FORMAT (22H1 THE SUM OF SQUARES = F15.3)
```

would cause the printing of (say)

```
| THE SUM OF SQUARES =     2791637.415
```

at the top of a new page.

10.6 Scanning of lists and FORMAT specifications

The scanning process may be summarised as follows. During execution of an input/output statement the associated FORMAT specification is scanned. Class 2 specifications are dealt with without reference to the input/output list. When a class 1 specification is encountered the list is referred to and if items remain to be transmitted the next is transmitted, according to the specification, and scanning of the FORMAT statement resumed. If the list is exhausted, then execution of the input/output statement ceases.

If the end of the FORMAT statement is reached before the list is exhausted, then the FORMAT statement is rescanned. The opening and closing parentheses of groups are thought of as paired at a certain level of depth: the parentheses enclosing the full format specification are paired at the zero level of depth and any groups within them at the first or deeper levels.

Rescan of a FORMAT statement starts a new record, and begins at the integer, if any, preceding the last opening bracket at the first level of depth, or at the beginning of the FORMAT specification if there are no brackets at the first level.

Example 9

A possible FORMAT specification to print the list

```
M,(I,(W(I,J),J=1,5),X(I),I=1,M)
```

starting on a new page according to the layout given in section 10.2, with a blank line between the value of M and the first row of array values, and a blank line after every 6 rows thereafter, would be

```
FORMAT (1H1,I2//6(I3,5E20.8,F15.8/))
```

Example 10

```
      READ 90,((TEXT(I,J),J=1,9),K(I),I=1,M)
   90 FORMAT (9A8,4X,I4)
```

These statements call for the characters from columns 1 to 72 of a set of cards to be read into an array TEXT, ignoring columns 73 to 76, and an integer to be read from columns 77 to 80 of each card.

Example 11

```
      PRINT 417, (TITLE(I),I=1,16),(RESULT(I),I=1,K)
  417 FORMAT (2(1X,8A8/)/(0P4F8.2,2P2E10.2))
```

These statements print two lines of title, followed by a blank line, followed by lines each containing 6 numbers, the first four in a fixed-point decimal form and the last two in floating point form, scaled up so that the mantissae lie in the range 10 to 100.

Exercises 10

1. A one-dimensional array ZZZ has N real elements ($N < 10^3$). Write statements to read the integer N from one card, and the elements of ZZZ punched in order on subsequent cards, 10 elements per card in adjacent 8-column fields.

2. The results of a program consist of two vectors P and Q, having 40 elements each. Values of P are less than one in absolute magnitude, and values of Q less than 1000. Write statements to print three columns: the first column a row number running from 1 to 40, the second column P and the third column Q . Print no more than 8 significant figures.

3. Non-zero elements only of an array BETA, dimensions 100 by 100, are punched four per card with integers identifying the row and column numbers. Write a loop of instructions to read all cards, specifying a suitable punching layout; devise a means of indicating the end of the pack of cards in order to terminate the loop.

4. Write statements to print out the elements of an array A , dimensions 30 by 20, as floating point numbers with a mantissa in the range 10 to 100 (i) printing 6 numbers per line (ii) printing 6 numbers per line, but starting each row of A on a new line (that is, starting a new line after every 20 elements). Retain 9 significant decimal figures.

5. An array PHI, dimensions 60 by 8, is considered as having 60 rows of 8 columns each. Write statements to output the elements of PHI as fixed point numbers, with two figures before the decimal point and five after, on 60 punched cards, each containing the elements of one row.

6. A program loop is to include the printing of two lines of results, preceded by one blank line, at each repetition of the loop, a typical output reading as follows:

```
ELEMENT  20, 31   LENGTH = 12.47 IN.   TYPE 3
STRESS =   400.7 LB/SQ.IN.
```

Write suitable output statements assuming that the values of length (< 100 inches), the type integer (0 to 9) and stress (< 10^5 lb/sq.in.) are to be found in the variables L, TYPE and STRESS respectively, and that two integers (each < 1000) identifying the element are to be found in N1, N2.

7. A program stores a case number (< 100) as the integer variable JOBNO and a title in a text array TITLE(8). Write statements to read the case number and title from one card.

The results of the program consist of values of WEIGHT (< 10^6 lb), HEIGHT (< 10^5 ft.), TIME (< 500 min.) and SPEED (< 2000 f.p.s.) stored as vectors each with N elements.

A page of printed results is to have at the top the word CASE, followed by the case number, followed by the title on the same line. Leaving two blank lines, the column headings WEIGHT (LB), HEIGHT (FT), TIME (MIN), SPEED (F.P.S.) are to be printed on the next line, followed by a further blank line before tabulating four parallel columns of actual values. These values are to be given to one decimal place in each case, with a blank line after every block of five lines. N is not so large that the results occupy more than one page altogether. Write statements to output the printed page.

SUGGESTED SOLUTIONS TO EXERCISES

Exercises 2

1. By forming elements of S in reverse, that is in the order $S_{i,q}$, $S_{i,q-1}$, ... unnecessary summation may be avoided.

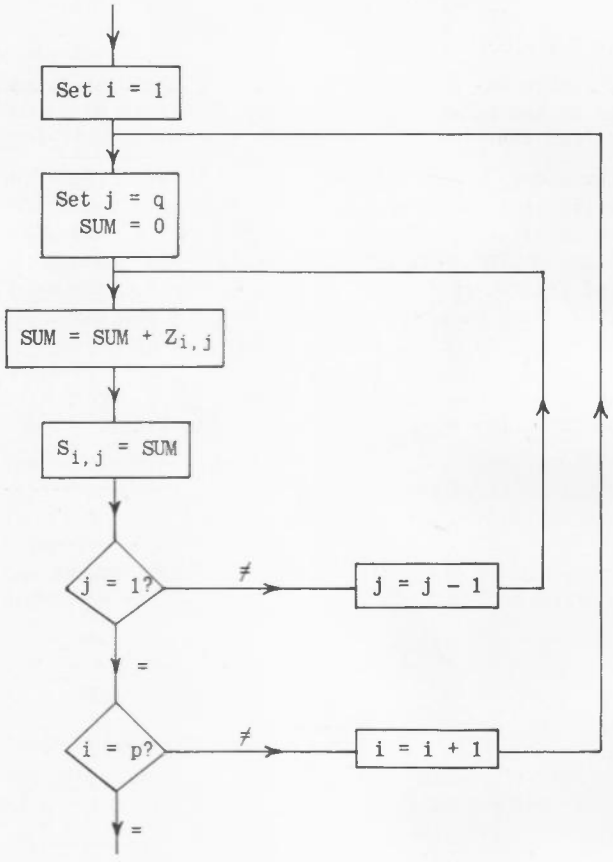


Fig. 1

2. A sketch of the graphs of bcosx and $1 - 3e^{-x}$ indicates that $f(x) = b\cos x - (1 - 3e^{-x})$ has roots between 0 and $\pi/2$, $3\pi/2$ and 2π , 2π and $5\pi/2$ and so on.

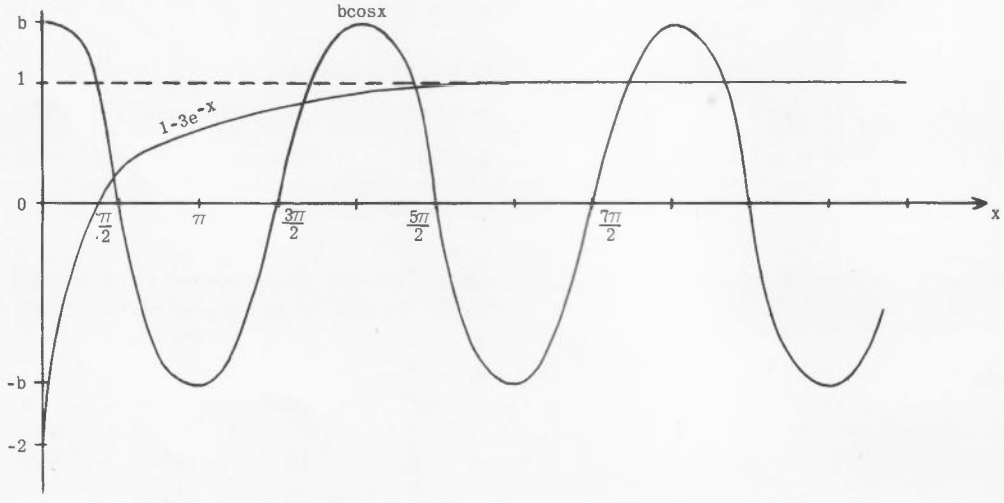


Fig. 2

Take as a first approximation to these roots the values $\pi/4$, $7\pi/4$, $9\pi/4$ and so on. If X is an approximation to a root, then a better approximation is $X + \Delta X$, where

$$\Delta X = -\frac{f(X)}{f'(X)}$$

In this case, $\Delta X = \dfrac{b\cos X - (1 - 3e^{-X})}{b\sin X + 3e^{-X}}$

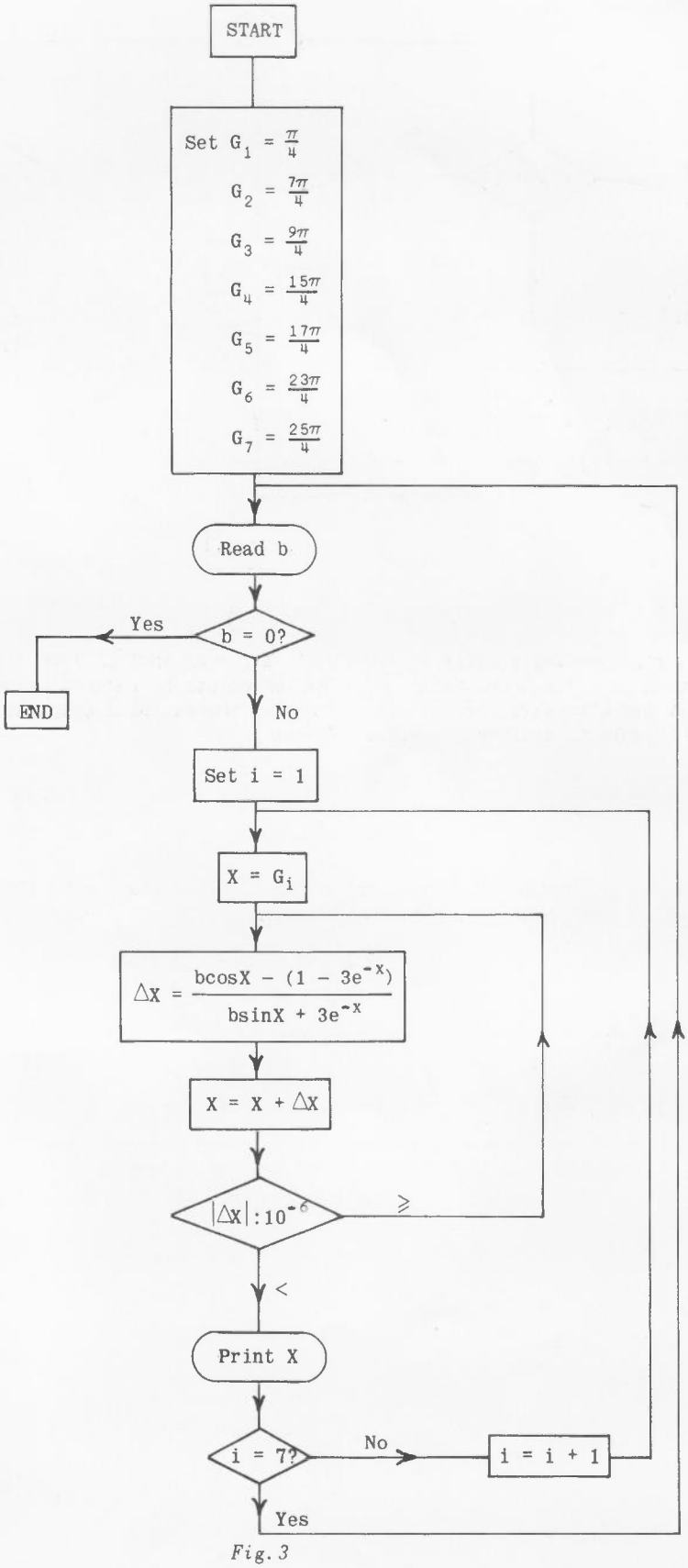


Fig. 3

3.

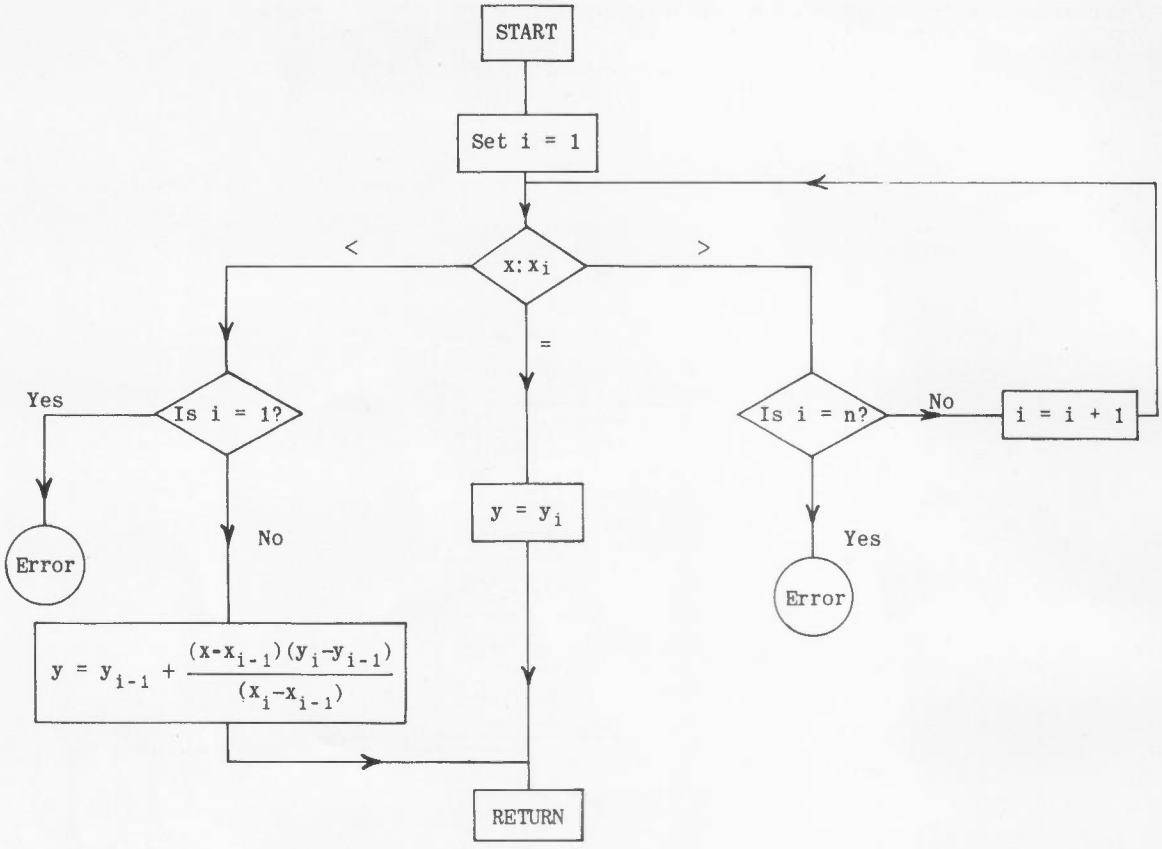


Fig. 4

This method of representing a graph is too simple for many applications. Repeated searching of a long list of ordinates is a time-consuming operation, which can be reduced by more sophisticated search procedures. It may also save time to store fewer ordinates for a graph and to use a more complicated interpolation formula to achieve comparable accuracy.

Exercises 3

1.

B(1)	Unacceptable	T-STEP	Unacceptable
A2.4	Unacceptable	DISTANCE	Unacceptable
WEIGHT	Real	JCOUNT	Integer
BBB	Real	△	Unacceptable
TEMP	Real	U2	Real
3K	Unacceptable	SUM	Real

2. 1200.0 147.3 0.0000000321 1256700.0

3.
(i) (A+4)/(2*B) or (A+4)/2/B

(ii) SQRTF(B+1.0/7.0)/7

(iii) 3*(A+B)

(iv) (B**2+2*D*E)**2

(v) A**(J+1)

(vi) (A+B)*(C+D)/(E+F)

(vii) A*X+B*X**2+C/D*X**3

or more economically

X*(A+X*(B+C/D*X))

(viii) (A/(A+1.5))**3.5

(ix) 0.5*LOGF(A/(B*(C+D)))

(x) SQRTF((A+B/(C+1))/D)

4.
(i) SUM = (N**2 * (N + 1)**2)/4

(ii) Q = (27*A**2*D - 9*A*B*C + 2*B**3)/(27*A**3)

(iii) T = (B - C)/(B + C)/TANF(A/2)

(iv) T = SQRTF((S-B)/S*(S-C)/(S-A))

(v) S = LOGF(1 + SQRTF(1 + X**2)) - LOGF(X)

(vi) INT = SINF(X)/(2*COSF(X)**2) + LOGF(1/COSF(X)+TANF(X))/2

(vii) AREA = 3.14159*(R+S)*SQRTF(H**2+(R-S)**2)

(viii) CG = H/4*(4*R-H)/(3*R-H)

(ix) Z,Y = EXPF(X)*COSF(2*X)

(x) Z = LOGF(ABSF(1/TANF(Y)))

(xi) U = SQRTF(X/3*TANF(3*X))

5.

(i)	A,B = C**2+1	Multiple assignment: variables should be separated by commas.
(ii)	A = -B/(2*C)	Two operators cannot be adjacent.
(iii)	A = (B+1)*(C+1)	Multiplication sign must be inserted.
(iv)	A = 4*LOGF(A+0.2)	Ditto.
(v)	A = SQRTF(B+2*(C+D))	Parenthesis missing.

(vi) Illegal and meaningless. An expression cannot appear on the left-hand side of an assignment statement.

(vii) π must appear in numerical form 3.14159... to the number of figures desired. The symbol π is not used with this meaning in the FORTRAN language.

6.
(i) X = -8.3333..

(ii) X = 0.75

(iii) N = -5

(iv) N = 3

(v) Y = X = 22.0

(vi) X = 42.25

(vii) N = -2

Exercises 5

1.
```
      REAL I,L
      READ 10,C,E,F,L,R
   10 FORMAT (5F10.0)
      P = 2*3.141593*F
      I = E/SQRTF(R**2+(P*L - 1/P/C)**2)
      PRINT 11, C,E,F,L,R,I
   11 FORMAT (6E18.5)
      CALL EXIT
      END
```

2.
```
      INTEGER P,SUM,SUMSQ
      READ 98,P
   98 FORMAT (I3)
      SUM = (P*(P+1))/2
      SUMSQ = (SUM*(2*P+1))/3
      PRINT 99, P,SUM,SUMSQ
   99 FORMAT (I6,I9,I12)
      CALL EXIT
      END
```

3.
```
      READ 12, B,C,ANGLEA
   12 FORMAT (3F10.0)
      A = SQRTF(B**2+C**2-2*B*C*COSF(ANGLEA))
      DELTA = B*C*SINF(ANGLEA)/2
      PUNCH 13, A,B,C
      PUNCH 13, DELTA
   13 FORMAT (3E20.8)
      CALL EXIT
      END
```

4.
```
      FAC = 3.141593/180
      READ 4, A,B,C
      A = A*FAC
      B = B*FAC
      C = C*FAC  πCONVERT A,B,C TO RADIANS
      EE = (A+B+C-3.141593)/2
      SINEE = SINF(EE)
      SA = SINF(A-EE)
      SB = SINF(B-EE)
      SC = SINF(C-EE)
      P = SQRTF(SINEE/(SA*SB*SC))
      AA = 2*ATANF(SA*P)/FAC
      BB = 2*ATANF(SB*P)/FAC
      CC = 2*ATANF(SC*P)/FAC
      PRINT 5, AA,BB,CC
    4 FORMAT (3F6.3)
    5 FORMAT (3F10.3)
      CALL EXIT
      END
```

Note: In Atlas FORTRAN the statement GO TO EXIT may be used in every case instead of CALL EXIT.

Exercises 6

1. (i)
```
IF(QNEW-QOLD-1E-8)50,5,5
```

(ii)
```
IF(J)1,,2
```

(iii)
```
IF(ABSF(Y)-ABSF(Z))9,,
      LOW = Z
      GO TO 10
    9 LOW = Y
   10 CONTINUE
```

(iv) (a)
```
      TOP = W
      IF(TOP.GE.X)1,
      TOP = X
    1 IF(TOP.GE.Y)2,
      TOP = Y
    2 IF(TOP.GE.Z)3,
      TOP = Z
    3 CONTINUE
```

(b)
```
TOP = AMAXF(W,X,Y,Z)
```

(v)
```
      IF(A1.LT.A2)11,10
   11 IF(A2.LT.A3) 9,10
```

Note: a single IF statement may be used, testing a relation of the form (A1.LT.A2.LT.A3). See section 9.

(vi)
```
      IF (I)1000,,
      IF (8-I)1000,,
      GO TO (12,13,14,13,14,13,15,15,15),I+1
```

2.
```
    7 READ 1,A,B,C
    1 FORMAT (3F10.0)
      IF (A),EXIT,
      D = B**2 - 4*A*C
      IF (D) 2,3,4
    2 R1,R2 = 0  ¤COMPLEX ROOTS
      GO TO 5
    3 R1,R2 = -B/2/A  ¤EQUAL ROOTS
      GO TO 5
    4 D = SQRTF(D)
      R1 = (-B+D)/2/A
      R2 = (-B-D)/2/A  ¤DISTINCT REAL ROOTS
    5 PRINT 6,R1,R2
    6 FORMAT (2E20.8)
      GO TO 7
      END
```

3.
```
      REAL IODD, IEVEN
      IODD = 1.0
      IEVEN = 3.14159265/4
      M = 1
      N = 2
    4 PRINT 200, M, IODD
      PRINT 200, N, IEVEN
      IF (N - 100),EXIT,
      M = M+2
      N = N+2
      IODD = IODD * (M-1)/M
      IEVEN = IEVEN * (N-1)/N
      GO TO 4
  200 FORMAT (I6,F10.6)
      END
```

4. (i)
```
      ZSQ = Z**2
      N = 3
      SUM, TERM = Z
   98 IF (ABSF(TERM) - 1E-8)99,,
      TERM = -TERM * ZSQ/N/(N-1)
      SUM = SUM + TERM
      N = N+2
      GO TO 98
   99 CONTINUE
```

(ii)
```
      W1 = H
      W2 = H**2
      W3 = THETA
      W4 = 2*THETA
      SUM = H*SINF(THETA)  ¤FIRST TERM
      DO 100 N = 3,39,2
      W1 = -W1*W2
      W3 = W3 + W4
      SUM = SUM + W1 * SINF(W3)/N**2
  100 CONTINUE
```

5.
```
      INTEGER X
      FOR X = 0,89,1
      TAN = TANF(X*.0174532925)
      T = TAN + TAN**3/3
      PRINT 1, X, T
    1 FORMAT (I4,F15.5)
      REPEAT
      GO TO EXIT
      END
```

Exercises 7

1.
```
      DIMENSION BETA (50), SN(50)
      FOR I = 1,50
      SN (I) = SINF (BETA(I) *.017453292)
      REPEAT
```

2.
```
      REAL K(100), K1(100), K2(100)
      DO 3 I = 1,N
      IF (K(I) -D),2,2
      K1(I) = 0
      K2(I) = K(I)
      GO TO 3
    2 K1(I) = K(I)
      K2(I) = 0
    3 CONTINUE
```

3.
```
      DIMENSION S(30), T(30), U(30)
      DO 2 I = 1,30
    2 S(I) = T(I) - U(I)
```

4.
```
      DIMENSION Y(301)
      SUM = 0
      FOR I = 1,3*N-2,3
      SUM = SUM+Y(I)+3*Y(I+1)+3*Y(I+2)+Y(I+3)
      REPEAT
      AREA = 0.375*H*SUM
```

5.
```
      DIMENSION A(40,50), ASQ(40,50)
      FOR I = 1,40
      FOR J = 1,50
      ASQ(I,J) = A(I,J)**2
      REPEAT
      REPEAT
```

6.
```
      REAL MASS (20,15), MMAX
      INTEGER ROWNO, COLNO
      MMAX = -1
      FOR I = 1,20
      FOR J = 1,15
      IF(ABSF(MASS(I,J)) - ABSF(MMAX))12,12,
      MMAX = MASS(I,J)
      ROWNO = I
      COLNO = J
   12 REPEAT
      REPEAT
```

7.
```
      DIMENSION X(200)
      FOR I = 1,N-1
      IF(X(I) - X(I+1)),99,99
      REPEAT
```

8.
```
      DIMENSION EXPR(31,19), CS(31)
      FOR I = 1,31
      EXPR(I,10),W=0.1*(I-1)
      CS(I) = COSF(W)
      REPEAT
      FOR J = 1,9
      E = 0.1*J
      ROOT = 1/SQRTF(1 - E**2)
      FOR I = 1,31
      EXPR(I,10-J) = ROOT*ACOSF((-E+CS(I))/(1-E*CS(I)))
      EXPR(I,10+J) = ROOT*ACOSF((E+CS(I))/(1+E*CS(I)))
      REPEAT
      REPEAT
```

9.
```
      DIMENSION A(50,50), B(50,50), C(50,50)
      FOR I1 = 1,I
      FOR K1 = 1,K
      C(I1,K1) = 0
      FOR J1 = 1,J
      C(I1,K1) = C(I1,K1) + A(I1,J1)*B(J1,K1)
      REPEAT
      REPEAT
      REPEAT
```

Exercises 8

1. (i)
```
      FUNCTION TANH(X)
      E = EXPF(-2*ABSF(X))
      E = (1-E)/(1+E)
      IF(X) 1,2,2
    1 TANH = -E
      RETURN
    2 TANH = E
      RETURN
      END
```

Note: Evaluation of tanh(x) in this way avoids any possibility of overflow which might otherwise occur when evaluating e^x for large values of x.

(ii)
```
      A = B**2 + 2*TANH(B/SQRTF(B**2 + C**2))
```

2. (i)
```
      FUNCTION SECH(X)
      SECH = 2*EXPF(-ABSF(X))/(1+EXPF(-ABSF(X))**2)
      RETURN
      END
```

(ii)
```
      REAL K(20)
      FOR I = 1,20
      W = 3.1415927*I/40  πVALUE OF πB/2A
      SUM = SECH(W)
      FOR N = 3,1000,2
      TERM = SECH(N*W)/N**2
      SUM = SUM+TERM
      IF(TERM - 1E-6)99,,
      REPEAT
   99 K(I) = 1-8*SUM/3.1415927**2
      REPEAT
```

Note: Within one statement, the FORTRAN compilers recognise and treat efficiently "common sub-expressions". Thus in the definition of SECH, the sub-expression EXPF (-ABSF(X)) will be recognised as appearing twice and instructions compiled in such a way as to call for its evaluation once only.

3. An outline of the analysis is as follows.
Expand K(x) as a series

$$K(x) = \frac{(\sin x - x\cos x)}{x^3}$$

$$= \frac{\left(x - \frac{x^3}{3!} + \frac{x^5}{5!} - \ldots.\right) - x\left(1 - \frac{x^2}{2!} + \frac{x^4}{4!} - \ldots.\right)}{x^3}$$

$$= \frac{1}{3} - \frac{x^2}{5.3!} + \frac{x^4}{7.5!} - \frac{x^6}{9.7!} + \ldots.$$

For small x , $\sin x \doteq x\cos x \doteq x$. Consequently, excessive cancellation occurs if the numerator of the original expression is formed by subtraction.

Consider x = 0.1 $\quad \sin x \doteq 0.1 - \frac{0.001}{6} = 0.9998333...$

$$x\cos x \doteq 0.1 - \frac{0.001}{2} = 0.9995$$

Upon subtraction, approximately 3 significant decimal figures are lost compared with the 11 significant figures to which x is expressed in an Atlas word. Consequently, for $|x| < 0.1$ use the series expansion. For x = 0.1 the term in x^6 has the value

$$\frac{10^{-6}}{45360} = \frac{1}{2} \times 10^{-10}$$

This and subsequent terms may be dropped.

A possible function routine is thus

```
      FUNCTION K(XX)
      REAL K
      X = XX
      IF(ABSF(X) - 0.1)1,1,
      K = (SINF(X)-X*COSF(X))/X**3
      RETURN
    1 K = (1 - X**2/10*(1-X**2/28))/3
      RETURN
      END
```

4. (i) SPECF(Y) = EXPF(Y) - SQRTF (1 - Y **2)

   (ii) A = SPECF (2 * T) + SINF (P * T)
        B = TANF(-SPECF(-N))

5.
```
      SUBROUTINE NORM (X, N)
      DIMENSION X(1)
      REAL MAX
      MAX = 0
      FOR I = 1, N
      IF (ABSF (X (I)) - ABSF (MAX))1,1,
      MAX = X(I)
    1 REPEAT
      FOR I = 1, N
      X(I) = X(I)/MAX
      REPEAT
      RETURN
      END
```

Notes: (a) It is assumed that at least one element of X is non-zero.
(b) The second statement could read DIMENSION X(N), N being an adjustable dimension. However, since X is a vector, the compiler makes no use of this statement except to note that X is the name of an array. The conventional dimension 1 is therefore permissible.

6.
```
      SUBROUTINE WHICH (ARRAY, N, MAX, I, J)
      DIMENSION ARRAY (100, 100)
      REAL MAX
      MAX = 0
      FOR II = 1, N - 1
      FOR JJ = II + 1, N
      IF (ABSF (ARRAY (II, JJ)) - ABSF (MAX)) 99,,
      MAX = ARRAY (II, JJ)
      I = II
      J = JJ
   99 REPEAT
      REPEAT
      RETURN
      END
```

Note: In any routine calling this routine, the name of an actual array substituted for the formal argument ARRAY must be declared in a dimension statement DIMENSION NAME (100, 100).

7.
```
      SUBROUTINE CHKSYM (A, N, ERROR)
      DIMENSION A(N, N)
      DO 98 I = 1, N - 1
      DO 99 J = I + 1, N
      IF (ABSF (A (I, J) - A (J, I) - 1E - 8),100,100
   98 CONTINUE
   99 CONTINUE
      RETURN
  100 CALL ERROR
      RETURN
      END
```

8. (i) either, in every routine

```
COMMON M, X, Y, A(100), B(10, 10), C(100, 50), D(10, 50)
```

or, in R1 COMMON M, X, Y, A(100)

in R2 COMMON M, X, Y, A(100), DUM(100), C(100, 50)

in R3 COMMON M, X, DUM(201), C(100, 50), D(10, 50)

in R4 COMMON M, DUM1(102), B(10, 10), DUM2(5000), D(10, 50)

(ii) in R1 PUBLIC M, X, Y, A(100)

in R2 PUBLIC M, X, A(100), C(100, 50)

in R3 PUBLIC X, C(100, 50), D(10, 50)

in R4 PUBLIC M, B(10, 10), D(10, 50)

Exercises 9

1. (i)
```
IF ((A1 .LT. A2 .LT. A3) .AND. (I .NE. 1)) 9,10
```

(ii) either

```
IF ((Y .LT. - 3) .OR. (Y .GE. 5)) 99,
```

or

```
IF ((Y .GE. -3) .AND. (Y .LT. 5)),99
```

2.
```
TEXT TITLE (4)
TITLE (1) = 6HCASE A
TITLE (2) = 6HCASE B
TITLE (3) = 6HCASE C
TITLE (4) = 8HNORESULT

PUNCH 1, TITLE (K)
1 FORMAT (A8)
```

Exercises 10

1. Assuming the input document to be on punched cards.

```
READ 99, N, (ZZZ (I), I = 1, N)
99 FORMAT (I3/(10F8.0))
```

The integer N is assumed to be punched (right adjusted) in the first three columns of the first card, and explicit decimal points punched in the data.

2.
```
PRINT 100, (I, P(I), Q(I), I = 1, 40)
100 FORMAT (I4, F15.8, F15.5)
```

3.
```
7 READ 101, (I, J, (BETA (I, J)), K = 1, 4), MARK
101 FORMAT (4(2I3, F10.0), 15X, I1)
IF (MARK), 7,
```

The last card of the pack has 1 punched in the last column. On other cards this column is blank (or zero). If the number of non-zero elements is not a multiple of 4, the last element must be repeated to make up the last card.

4. (i)
```
PRINT 102, A
102 FORMAT (2P6E20.7)
```

(ii)
```
PRINT 102, A
102 FORMAT (2P3(6E20.7/), 2E20.7)
```

5.
```
PUNCH 103,((PHI(I,J), J = 1,8), I = 1,60)
103 FORMAT (8F10.5)
```

6.
```
PRINT 104, N1, N2, L, TYPE, STRESS
104 FORMAT (8HOELEMENT I4, 1H, I3, 3X8HLENGTH = F6.2, 4H IN. 3X4HTYPE I2/9H STRESS = F8.1, 10H LB/SQ.IN.)
```

```
7.    READ 105, JOBNO,(TITLE(I), I = 1,8)
  105 FORMAT(I2, 8A8)

      PRINT 106, JOBNO,(TITLE(I), I = 1,8)
  106 FORMAT(5H1CASE I3, 5X, 8A8///
             45H WEIGHT(LB) HEIGHT(FT) TIME(MIN) SPEED(F.P.S)/)
      PRINT 107,(WEIGHT(I),HEIGHT(I),TIME(I),SPEED(I), I = 1,N)
  107 FORMAT(5(F10.1, F11.1, F10.1, F12.1/))
```

MTD/dmh
31.10.63