THE STUDENT MANUAL FOR PROGRAMMING IN FORTRAN IV

A Thesis

Presented to

the Faculty of the Department of Mathematics

Kansas State Teachers College at Emporia

In Partial Fulfillment

of the Requirements for the Degree

Master of Arts

by

Mok Tokko

August 1968

Approved for the Major Department

Approved for the Graduate Council

ACKNOWLEDGEMENT

The writer wishes to thank Professor Lester E. Laird who gave assistance and inspiration in the preparation of this paper.

TABLE OF CONTENTS

CHAPTER PAGE

LIST OF FIGURES

CHAPTER I

INTRODUCTION

This manual is written for the students who are enrolled in the Mathematical Programming course offered by the Mathematics Department at the Kansas State Teachers College at Emporia. It is written for the student who has no knowledge of computer programming and no more background in mathematics than the one acquired in the first two years in college.

The primary purpose of this manual is to give a comprehensive analysis of the language used in the FORTRAN IV programming with numerous examples and illustrations. In the Mathematical Programming course at this college, a large portion of the class periods are devoted to learning how to solve problems, thus leaving little time to learn the language of the FORTRAN IV programming.

It is hoped that this manual will be easy to read and that the student will acquire the basic knowledge of FORTRAN IV programming in a short period of time. The sample programs are written short to better illustrate the subject in discussion. Longer programs illustrating the methods of solving mathematical problems are included in the appendix.

This manual discusses only the topics which are essential to a programmer in the field of mathematics. The detailed discussion of physical components of the computer is not included. This manual is

written specifically for programming in FORTRAN IV for IBM 1401 which is available at the present time at the Data Processing Center at the Teachers College. The exact and complete details not included in this manual can be found in the manual published by the International Business Corporation, FORTRAN IV Language Specification, File No. GENL-25, Form C24-3322-3.

CHAPTER II

COMPUTER AND COMPUTER PROGRAMMING

To a mathematician, a computer is a machine that solves mathematical problems in an incredibly short period of time. It can find a square root of a number, add complex numbers, multiply matrices, solve differential equations, and almost any type of problems a mathematician encounters. But, what a computer really can do is to add two numbers. It is a giant adding machine with one added feature which makes the computer so versatile. It has the ability to store information.

In reality, a computer does not solve those complicated problems. It is the mathematician who devised the technique of solving complicated mathematical problems by simply adding two numbers. In fact, a computer cannot even multiply two numbers. In solving a simple problem such as multiplying eight by five, the computer adds eight five times. Time is not a problem to a computer. A computer such as the IBM 1401 can perform thousands of additions per second.

I. FUNCTIONAL COMPONENTS OF A COMPUTER

In general, all computers can be divided into four functional units; the input, the output, the memory, and the control.

All information is entered into the computer by way of the input unit. The information is usually recorded on a magnetic tape or on the standardized card. The student in the programming course will use IBM cards.

The information obtained by the computer reaches the programmer

by means of the output unit. There are several output devices. The results can be recorded on a magnetic tape, punched on IBM cards, or printed on paper. The student will normally receive the result of the program printed on paper.

One of the most fascinating aspects of a computer is its ability to store information. The memory unit enables the computer to "remember." Once the information is stored in the memory unit, it can be brought out of the unit without destroying the record in the memory unit. The information can be stored in the core storage within the computer, or on the magnetic tape, or on the disk storage.

The control unit of a computer is considered as the "brain." This control unit receives information from the memory, interprets the information, performs the necessary operation, and stores the results back into the memory unit.

II. PROGRAMMING

Computer programming is writing a sequence of instructions for a computer to carry out in the solution of a problem and the written instructions are called a program. With the early computers the instructions were written in machine language which is a basic numeric code the computer could accept and execute. Writing a program in the machine language was a tedious job. It was long and time consuming. Consequently, there were many opportunities to make mistakes. Then, one developed the FORTRAN system of writing a program. The name stands for FORmula TRANslation. This system is human or problem oriented language rather than machine oriented

language. By using the FORTRAN language, the time and effort for both writing and correcting a program is greatly reduced.

The instructions written in FORTRAN language resembles the expressions written in mathematics. These instructions are fed into the computer and the computer translates them into the machine language. This translating system is known as a compiler. The compiler program is written by an expert programmer.

One disadvantage of using a translating system is that the system may not make as efficient use of the computer as an expert programmer writing machine language. An expert programmer can take full advantage of a computer's capabilities and write the program to use a minimum amount of computer time. However, the use of a compiler to write the machine language program saves much programming time.

III. EQUIPMENT

This is a list of the equipment available at the present time at the Kansas State Teachers College Data Processing Center.

IBM 1401 Processing Unit. This is the control unit of the computer. It has an internal memory unit of 12,000 positions of core storage, and all the computation is performed in this unit.

IBM 1406 Storage. This memory unit has 4,000 positions of core storage, thus giving the processing unit total core storage of 16,000 positions.

IBM 1311 Disk Storage Drive. At present, there are three of these memory units at the Data Processing Center but only one is used when processing a program written in FORTRAN. Each of these disks has 2,000,000 storage positions.

IBM Card Read-Punch. This is the primary input unit of the computer. The information punched on cards is transferred to the core storage through this unit. It also punches out cards containing output data if this type of output is desired.

IBM 1403 Printer. Most of the output and other communication from the computer to the programmer is made through this high speed printer.

IBM 407 Accounting Machine. The student in the programming course will use this machine for the purpose of printing the information punched on cards.

IV. DEFINITION OF KEY TERMS

In order to be able to read this manual satisfactorily the reader is urged to become familiar with the meaning of following terms.

Compiler. This is a program written by an expert programmer to translate the program written in FORTRAN language into machine language. The process is called compilation.

Program. A program is a sequence of instructions which a computer is to carry out in the solution of a problem.

Source Program. A program written in FORTRAN symbolic language.

Executable Statement. A program statement which contains an instruction for the computer to "do" something.

Non-Executable Statement. A program statement which contains no executable instruction.

Field, Record, and File. A field is the smallest division of data and refers to a single item or number. A record consists of one or more fields. For example, a program which adds two four by four matrices may contain eight data cards. Each card contains four numbers, the elements of a row. The complete set of data cards, eight in this case, is a file, each card is a record, and each number on a card is a field. In other words, this file contains eight records and each record contains four fields.

Debugging. This term refers to a process of finding mistakes by a programmer in a program.

Control Card. A control card contains the necessary information for the operation of a computer. For example, FORTRAN RUN card instructs the computer to start translating a source program, and $EXECUTION card instructs the computer to start processing the program which has been translated into machine language.

Type. The type refers to the kind of variables or numbers used in a program. There are two types of variables and numbers; a real type and integer type.

CHAPTER III

CONSTANTS, VARIABLES, SUBSCRIPTS, AND EXPRESSIONS

The meaning of the terms, constants, variables, and subscripts is the same in FORTRAN programming as in mathematics. There are, however, some limitations in the use of these terms in programming.

I. CONSTANTS

A constant is a real number, positive, negative, or zero, which appears in a program statement. There are two types of constants; integer constants and real constants.

Integer Constants.

An integer constant is an integer, positive, negative, or zero that appears in a program statement. An integer constant is written without a decimal point. In elementary mathematics the numbers 2 and 2.0 are sometimes used interchangeably, but in FORTRAN programming, this is not allowed. The following are examples of integer constants.

0

1

-27

35029

If a programmer wishes to use an integer constant containing six digits or more, the largest number of digits desired to be used must be indicated to the computer by using a special program statement.

$INTEGER SIZE = nn

The instruction for preparing this control card is on page 89 . The maximum number of digits an integer constant can have is twenty. If $INTEGER SIZE = nn card is not provided, only a constant with five or less digits may be used.

Real Constants

In FORTRAN programming a real constant is a real number, positive, negative, or zero, and must contain a decimal point. A real constant may be written with an exponent. The following are examples of real constants.

0.

1.

-2.04

700.

5.12E+02

In expressing a real constant in an exponential form, a number with a decimal point is followed by the letter E, a plus or minus sign, and a two-digit integer. The following are the examples of real constants in an exponential form.

5.7E+02 for 5.7×10^2

-5.7E+02 for -5.7×10^2

2.3E-03 for 2.3×10^{-3}

.4E3 for $.4 \times 10^3$

4.1E 00 for 4.1×10^0 or 2.3

-3.45E-02 for -3.45×10^{-2}

If the space immediately following E is left blank, the computer assumes the exponent to be positive. If an integer appears in the space reserved for the sign, the computer also assumes the exponent to be positive.

The number of significant digits a real constant can have depends upon the computer. If a real constant contains more than eight significant digits, this must be indicated to the computer using a special program statement.

$REAL SIZE = nn

The instruction for preparing this control card is on page 89. A real constant can have at most twenty significant digits.

II. VARIABLES

A variable is a symbol that represents an element of a set just as in elementary algebra. For example, in expressing the area of a circle, $A = \pi r^2$, r is a variable and π is a constant. In algebra a single letter is used as a variable. In FORTRAN language, however, a variable may be written as one letter or up to six letters or digits. A variable can be written using both letters and numbers, but the first character must be a letter. The first letter must be chosen carefully because it indicates whether the number represented by the variable is an integer constant or a real constant.

Integer Variable. An integer variable is a variable for which only an integer constant may be substituted. The first letter of a variable must be I, J, K, L, M, or N. The following are examples of

integer variables.

I, J, MAX, JOB, IDIOT, N3.

Real Variable. A real variable is a variable for which only a real constant may be substituted. The first letter of a real variable must be any letter except I, J, K, L, M, and N. The following are examples of real variables.

X, Y, ALPHA, ROOT, SUM, PROD, C1.

Subscripted Variables. In FORTRAN programming a subscript cannot be written in the customary way. A subscripted variable consists of a variable name followed by parentheses enclosing one, two, or three subscripts separated by commas. The first letter of the subscripted variable must be chosen with the consideration to the type of variable involved. All the elements in one array must be of the same type, either all integer constants or all real constants.

Example. The matrix A below consists of five numbers and the dimension of this matrix is said to be one, since there is only one column of numbers. The matrix B consists of six numbers which are arranged in two rows and three columns. The dimension of this matrix is said to be two, since the matrix consists of two or more rows and two or more columns.

$$A = \begin{pmatrix} 4 \\ 3 \\ 5 \\ 2 \\ 8 \end{pmatrix} \qquad B = \begin{pmatrix} 2.3 & 3.4 & 5.1 \\ 7.0 & 2.9 & 0.4 \end{pmatrix}$$

The matrix A, which consists of all integers, can be represented with a subscripted variable, M(I), where I varies from 1 to 5. The elements of A are individually represented as follows.

M(1) has the value 4.

M(2) has the value 3.

M(3) has the value 5.

M(4) has the value 2.

M(5) has the value 8.

It should be noted that the subscripted variable must be an integer variable.

The matrix B can be represented with a real subscripted variable, A(I,J), since it contains all real numbers. I varies from 1 to 2 and J varies from 1 to 3. The elements of B are represented as follows.

A(1,1) has the value 2.3.

A(1,2) has the value 3.4.

A(1,3) has the value 5.1.

A(2,1) has the value 7.0.

A(2,2) has the value 2.9.

A(2,3) has the value 0.4.

Form of Subscripts. A subscript can take only one of the following forms. V is an unsigned, nonsubscripted integer variable and C and C' are unsigned integer constants. It should be noted that M(2+I) and M(I*2) are not acceptable, but M(I+2) and M(2*I) are acceptable.

General form	Example
C	1 as in M(1)
V	I as in M(I)
V+C	I+2 as in M(I+2)
V-C	I-1 as in M(I-1)
C*V	2*I as in M(2*I)
C*V+C'	2*I+1 as in M(2*I+1)
C*V-C'	2*I-1 as in M(2*I-1)

Arithmetic Expressions

The function of an arithmetic expression is to produce a single numerical value equivalent to the value of the expression by performing a certain operation or operations upon the constants or variable names.

Symbols	Meaning	
+	Addition	as A+B
-	Subtraction	as A-B
/	Division	as A/B
*	Multiplication	as A*B
**	Exponentiation	as A**B (A to the B power)

Order of Operation.

Parentheses are used, as in algebra, in expressions to specify the order in which the expression is to be evaluated. Expressions are evaluated from left to right. If parentheses are omitted, the order of computation is as follows.

1. Function computation and substitution.
2. Exponentiation.
3. Multiplication and division.
4. Addition and subtraction.

Examples. A**2.0*B/C + D + 3.0 will be treated as in algebra as $\frac{A^2B}{C} + D + 3.0$.

(A-B*C/D+1.0)**2.0 will be calculated as $(A - \frac{BC}{D} + 1.0)^2$.

A/B+C will be treated as $\frac{A}{B} + C$, not as $\frac{A}{B+C}$ as one might expect.

((A-B)/(C+1.0))**2.0 will be calculated as $(\frac{A-B}{C+1})^2$.

The student is urged to use parentheses liberally to make sure the operations are performed in the desired order. The expressions shown below are improper. The first expression must be written as A - B and the second expression must be written as A/(-B).

```
A + -B
A/-B
```

Invalid Expressions. It must be noted that an arithmetic expression must all be either integer values or all real values. In other words, an integer cannot be added to or multiplied by a real number, or

vice versa. There is, however, one exception. A real number can be raised to an integer power.

ALPHA + 1	(An integer constant cannot be added to a real variable)
A*I	(An integer variable cannot be multiplied by a real variable)
I**2.0	(An integer variable cannot be raised to a real power)
N/2.0	(An integer variable cannot be divided by a real number)

Relational Operation Symbols

The relational operation symbols are listed below together with their meanings. These symbols are used primarily with logical IF statements. The periods are part of the symbol.

Symbols	Meaning	Mathematical notation
.GT.	Greater than	$>$
.GE.	Greater than or equal to	$\geq$
.LT.	Less than	$<$
.LE.	Less than or equal to	$\leq$
.EQ.	Equal to	$=$
.NE.	Not equal to	$\neq$

CHAPTER IV

PROGRAMMING PROCEDURE

It is very important to follow a proper procedure if a student is to write a good program. This chapter is devoted to the discussion of what a program consists of and how to write a program. It contains some FORTRAN statements that are yet to be explained but these are used so that the sample program is complete.

I. PROGRAM

In general, a program consists of six major parts; (1) beginning control cards; (2) the input instruction; (3) computations; (4) the output instructions; (5) ending control cards; and (6) data cards.

Beginning Control Cards. These cards contain the information necessary for a computer to function properly. For instance, the FORTRAN RUN card is necessary to start the compiler. The $NO MULTIPLY DIVIDE card informs the compiler that the computer has no multiplying device.

The Input Instruction. This part instructs the computer to read the data and any other information necessary for solving a problem. This is accomplished with a READ statement. A program may contain more than one READ statement. The data read into the computer with a READ statement is automatically stored in the core storage.

Computations. This part may be considered as the main part of a program. It consists of one or more arithmetic statements. When the computation is long or complex, this part may be divided into

subdivisions. These subdivisions are called subprograms and a detailed discussion is given in Chapter VIII.

The Output Instruction. This instructs the computer to write the data obtained from the computation. This is accomplished with a WRITE statement. A program may contain one or more WRITE statements and the output data is usually printed on paper.

Ending Control Cards. These control cards accomplish three things; (1) they inform the compiler of the end of the program; (2) they instruct the computer to store the translation of the program in the core storage; and (3) they instruct the computer to start processing the data.

Data Cards. Finally, data cards are placed at the very end of each program.

II. FLOW CHART

A flow chart is a graphic representation of the method used to solve a problem. The purpose of a flow chart is to help the reader as well as the writer to understand the logic of a complicated problem. Preparing a flow chart before writing a program helps the student to write a better program. A flow chart also helps the student to check the program after it is written.

A flow chart consists of various geometric figures connected by lines. The following is the description of various flow charting symbols which are commonly used in computer programming.

A rectangle is used to represent computations. The statement number can be written in the upper left hand corner.

A trapezoid is used for all input and output.

A diamond represents a decision point in a program, such as IF statement.

A small circle may be used to connect various parts of a flow chart. The circle may contain a statement number.

A hexagon is used to represent a subprogram, a Library Function, and FUNCTION or SUBROUTINE subprograms.

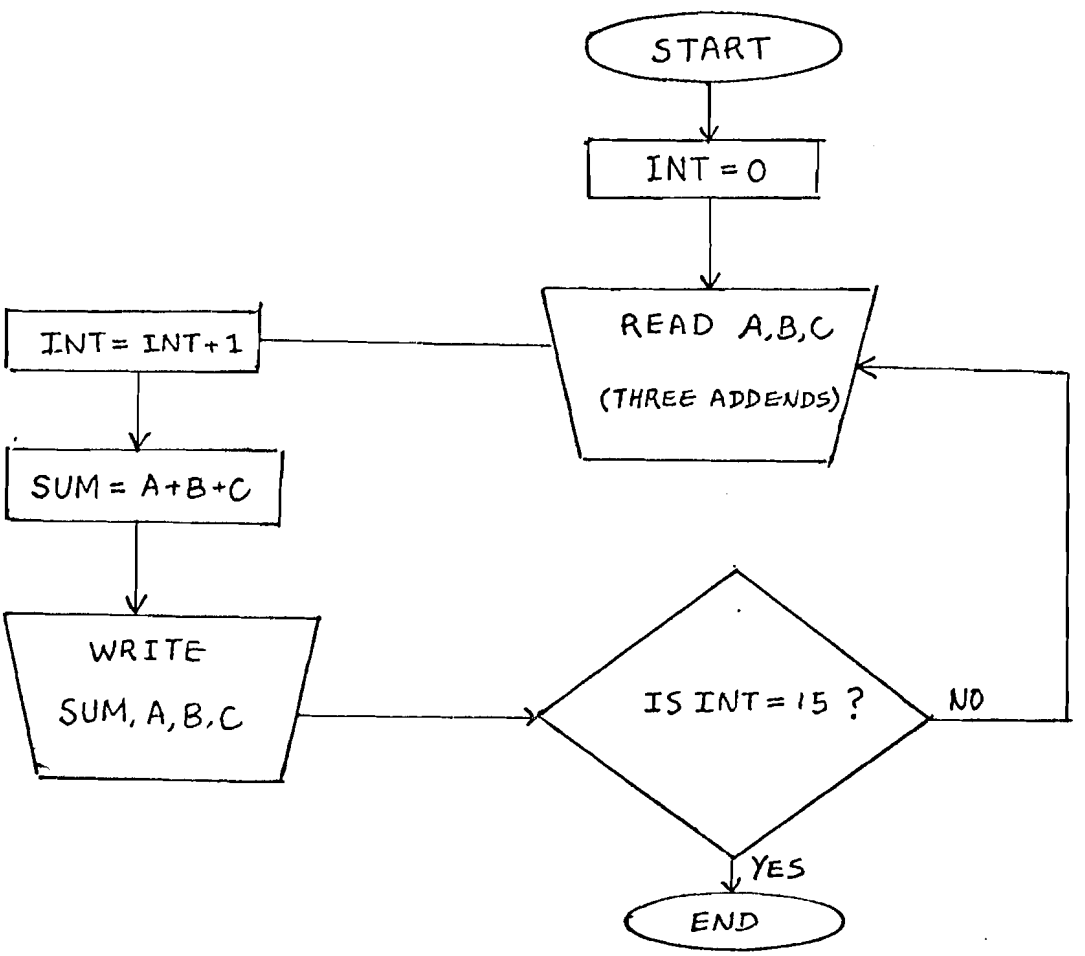


FIGURE 1

A FLOW CHART ILLUSTRATING
THE PROGRAM WHICH ADDS THREE NUMBERS

Sample Program. This program is for finding the sum of three numbers and it illustrates how a program is written from a flow chart. Three numbers whose sum is to be found will be on a data card. The computer is instructed to read the numbers, label the first number A, the second B, and the third C, and store them. The sum of the three numbers is found, labled SUM, and stored. There will be fifteen data cards. The variable name INT has the value 1 when the first data card is processed. The value of INT is, then, increased by one each time an additional data card is processed. When the value of INT is 15, that is, when the fifteenth card is processed, the program is terminated.

```
      FORTRAN  RUN
      $NO MULTIPLY DIVIDE
      $NO DICTIONARY
      INT = 0
4     READ (1,1) A, B, C
1     FORMAT (3F8.2)
      SUM = A + B + C
      INT = INT + 1
      WRITE (3,2) A, B, C, SUM
2     FORMAT (3F8.2, F12.2)
      IF (INT.NE.15) GO TO 4
      END
      LOADER  RUN
      $EXECUTION
```

Input data:

```
2.      3.      5.
1.      4.      7.
3.4     5.2     4.8
10.     12.     31.
```

Output:

2.00	3.00	5.00	10.00
1.00	4.00	7.00	12.00
3.40	5.20	4.80	13.40
10.00	12.00	31.00	53.00

III. WRITING PROGRAM

To a mathematician, programming is solving a problem using a computer. Therefore, it is necessary to know how to solve the problem without a computer. The problem must be analyzed step by step. The student must, then, decide which method or algorithm to use to solve the problem.

The program is written in FORTRAN IV language. Each statement must be exact and accurate. Each statement, then, is punched on a standardized IBM card.

Control Cards. The following three control cards must precede every source program. They are;

```
FORTRAN  RUN
$NO MULTIPLY DIVIDE
$NO DICTIONARY
```

The FORTRAN RUN card must always be the first card of every program and subprogram.

The following two control cards must follow every program. They are;

```
LOADER  RUN
$EXECUTION
```

The $EXECUTION card must be the last card of every program. Data cards are placed following the $EXECUTION card.

Name Card. It is recommended that a name card be included with every program so that the computer operator can identify the source deck. The name card must have C punched in the first column. The name may be punched in any of the remaining columns. The name card should be placed immediately following the first three control cards.

Comment Cards. The use of comment cards helps both the writer and the reader to understand the program. A comment card can be prepared by simply punching the letter C in the first column. Any explanatory comments can be punched in the columns 2-72. The student is urged to make liberal use of comment cards in the program.

The comment card can be placed anywhere in the program. When

the compiler encounters a C in the first column, it ignores the remainder of the card, but the comment punched on the card will appear in the program listing printed by the computer. Every program should have a comment card containing the title of the program and it should be placed immediately following the name card.

IV. PREPARATION OF SOURCE DECK

There are eighty columns on an IBM card for the FORTRAN IV programmer and they are divided into four groups.

Columns 1-5. The first five columns are reserved for a statement number. This number must be unsigned and five digits or less. The numeral 0 may not be used as a statement number and all statement numbers must be unique. However, they need not be in any sequence. The statement number 25 can be written anywhere in the first five columns, provided that 2 appears first.

Column 6. If a statement is too long to write on one card, additional cards may be used up to nine cards. In this case, all additional cards must contain a character in the column 6; any letter of the alphabet or any number between 1 and 9 inclusive. The numbers or the letters need not be arranged in sequence if more than one additional card is used. The numeral 0 should not be used. This method is often used in a FORMAT statement which prints headings or titles.

Columns 7-72. The actual FORTRAN statements are written in these columns.

Columns 73-80. These columns are not processed by the compiler

and can be used for identification of the card.

V. LISTING PROGRAM

The source deck should be processed through the IBM 407, the accounting machine. This machine transfers the information punched on cards into a readable document. The student is strongly urged to check for possible key-punch errors.

CHAPTER V

GENERAL FORTRAN STATEMENT

Three types of FORTRAN statements are discussed in this chapter; (1) arithmetic statements which instructs a computer to perform an arithmetic computation; (2) control statements which indicate the order in which other statements are performed; and (3) specification statements which provides a computor the necessary information for storing and handling of data.

I. THE ARITHMETIC STATEMENT

A FORTRAN arithmetic statement closely resembles a conventional algebraic formula or an equation, except that the equal sign (=) does not stand for equivalence. The general form is;

```
a = b
```

where a is a single variable name and b is either a single variable name or an arithmetic expression. The following are examples of arithmetic statements.

```
A = B
SUM = X + Y
ROOT1 = -B + SQRT (B**2 - 4.*A*C)
```

To a computer = means, "find a single numerical value equivalent to the expression on the right of the = sign and store it in a location to be referred to by the variable name on the left side

of the = sign. The sample statement, SUM = X + Y, causes the computer to add the number represented by X and the number represented by Y and store the sum which is to be referred to by a variable name SUM.

In an arithmetic statement the variable name on the left of = sign determines the type of the number obtained from the expression on the right side of = sign. The expression cannot be of mixed types. When an arithmetic statement, J = B, is executed, since J is an integer variable and B is a real variable, the fractional part of the number represented by B is discarded and resulting number, which is an integer, is stored and given the name J.

When an arithmetic statement, A = L + M, is executed, two integer constants represented by L and M are added, the single resulting integer is converted to a real constant, and this number is given the name A. If L = 4000 and M = 500 in the statement above, then $.45 \times 10^4$ is stored in A.

Sample Program. The following program is for finding the sum of two numbers and illustrates the use of real variables and an arithmetic statement.

The numbers whose sum is to be found will be on a data card. The computer is instructed to read the numbers, label the first number X and the second Y and store them. The sum of X and Y is found, labeled SUM, and stored. Then, finally, the computer is instructed to print X, Y, and the SUM. This program is written to process only one data card.

```
      FORTRAN  RUN
      $NO MULTIPLY DIVIDE
      $NO DICTIONARY
      READ (1,10) X, Y
   10 FORMAT (2F8.2)
      SUM = X + Y
      WRITE (3,11) X, Y, SUM
   11 FORMAT (3F8.2)
      END
      LOADER  RUN
      $EXECUTION
```

Input data:

```
3.00     4.00
```

Ouput:

```
3.00     4.00     7.00
```

Every source program must contain three control cards at the beginning of the program and two control cards at the end of the program as shown in the sample program above. Further information concerning these cards can be found on page 88.

The first number inside the parentheses of the READ statement is a symbolic name of the input unit associated with reading of cards and this number is always 1. The second number is a statement number of the FORMAT statement that tells the machine where and in what form

the numbers are to be stored in the memory. The two numbers on the data card are read in according to the FORMAT statement whose number is 10. All numbers must be read into the computer according to some FORMAT specification. Further information concerning FORMAT statements can be found on page 54.

The statement, SUM = X + Y, is an arithmetic statement whose function is to perform computation. An arithmetic statement always contains an = sign.

The WRITE statement instructs the computer to write the result of computation and any other information the programmer may wish to have. The first number inside the parentheses in the WRITE statement is a symbolic name of an output unit associated with the printing of data and this number is always 3 for the printer. The second number is a statement number of a FORMAT statement. The three numbers are printed according to the FORMAT statement whose number is 11. All output numbers must be printed according to some FORMAT specification.

Every program must end with an END statement. This statement instructs the compiler to stop translating.

II. THE CONTROL STATEMENT

There are, in general, three types of control statements; (1) GO TO statement; (2) IF statement; and (3) DO statement.

The Unconditional GO TO Statement

In a program, sometimes it becomes necessary to direct the flow of a program to a statement other than the one immediately

following. The GO TO statement makes it possible to go back to a statement which has already been executed or skip one or more statements. The general form is;

GO TO n

where n is a statement number of an executable statement.

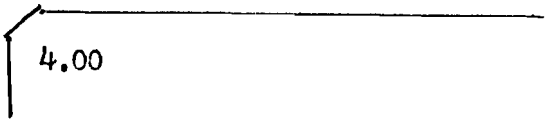
Sample Program. This program is written for finding the reciprocal of a number and it illustrates the use of the unconditional GO TO statement.

The computer is instructed to read a number, label it R, store it, and check it to see if the number is zero. If the number is zero, the computer is instructed to skip the process of finding the reciprocal. Even the computer cannot divide by zero. The number zero is labeled Y and the computer is instructed to write R and Y. If the number is not zero, the reciprocal is found and the computer writes the numbers R and Y which is the reciprocal of the number.

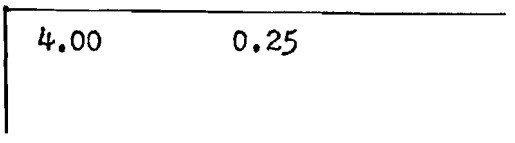
```
    FORTRAN  RUN
    $NO MULTIPLY DIVIDE
    $NO DICTIONARY
    READ (1,11) R
11  FORMAT (F6.2)
    IF (X - 0.) 2, 4, 2
 4  Y = 0.
    GO TO 7
 2  Y = 1./R
```

```
 7  WRITE (3,12) R, Y
12  FORMAT (F6.2, F10.2)
    END
    LOADER  RUN
    $EXECUTION
```

Input data:



Output:



Note. The unconditional GO TO statement should not be used at the end of a program for the purpose of repeating the computational process when more than one data cards are used. The following program is fine if it is processed through the computer by itself. This program will continue processing the data cards until there are no more cards left in the input card hopper. However, the computer operator at the Data Processing Center processes several programs at a time. After the last card of this particular program is processed, the computer will read the cards of the next program as data cards. If a repetition is desired, the student should use a DO statement.

The following is a sample program which contains an unconditional GO TO statement at the end of the program.

```
      FORTRAN  RUN
      $NO MULTIPLY DIVIDE
      $NO DICTIONARY
    5 READ (1,21) S, T
   21 FORMAT (2F6.2)
      QUOT = S/T
      WRITE (3,22) S, T, QUOT
   22 FORMAT ( 2F6.2, F10.2)
      GO TO 5
      END
      LOADER  RUN
      $EXECUTION
```

The Computed GO TO Statement

This statement causes the computer to be transferred backward or foreward to the statement n_1, n_2, . . . , n_m, depending on the value of i. The general form is;

GO TO (n_1, n_2, . . . , n_m), i

where i is a nonsubscripted integer variable and it can represent a number between 1 and 9 inclusive, and n is a statement number.

Since the limit of i is 9, m cannot be larger than 9. In other words, the parentheses cannot contain more than nine statement numbers.

Illustration. In the statement below, if the value of K is 1, the statement 4 will be executed. If the value of K is 2, the statement 13 is executed, and if the value of K is 3, the statement 28 is

executed. Thus K is used as a code number.

```
      GO TO (4, 13, 28), K
```

Sample Program. The following program is for finding the square root of a number or the square of the number. The computer is instructed to read two numbers, label the first number X and the second K and store them. If K is 1, the square of the number is found and given the name Y. If the value of K is 2, the square root of the number is found and given the name Y. The computer then is instructed to write X, K, and Y.

```
      FORTRAN  RUN
      $NO MULTIPLY DIVIDE
      $NO DICTIONARY
      READ (1,31) X, K
31    FORMAT (F6.2, I4)
      GO TO (42, 41), K
41    Y = SQRT (X)
      GO TO 61
42    Y = X**2
61    WRITE (3,32) X, K, Y
32    FORMAT (F6.2, I4, F8.2)
      END
      LOADER  RUN
      $EXECUTION
```

Input data:

```
12.00     1
```

Output;

```
 12.00     1        144.00
```

The Logical IF Statement

The purpose of the logical IF statement is to make a decision. Depending upon the value of a logical expression, (a), choice is made, deciding which one of two statements to execute next. The general form is;

```
IF (a) s
```

If the logical expression, (a), is false, the computer executes the next sequential statement. If (a) is true, the computer executes the s, where s is a statement, not a statement number. GO TO statement is often used along with the logical IF statement. In the partial program below, if I = 3 and L = 4, then (I.GT.L) is false and the statement immediately following the IF statement is executed. But, if I = 4 and L = 3, then (I.GT.L) is true and the statement 24 is executed next. Thus, the second statement is bypassed.

```
      .
      .
      .
      IF (I.GT.L) GO TO 24
      X = A - B
   24 Y = A + B
      .
      .
      .
```

Sample Program. This program finds the average of three numbers and illustrates the use of a logical IF statement.

The computer is instructed to read three numbers on a data card, label the first A, the second B, and the third C, and store them. Then, the three numbers are added and the sum is divided by three to find the average. The resulting number is labeled AVEG and stored. The computer is instructed to print three numbers and the average, AVEG. The value of INT is initially zero and each time a data card is processed, one is added to the value of INT, thus at the end of processing the fifth card, the numerical value of INT is 5 and the program comes to an end. As long as the value of INT is not 5, the loop is repeated. Naturally, this program must be followed by five data cards.

```
      FORTRAN  RUN
      $NO MULTIPLY DIVIDE
      $NO DICTIONARY
12    INT = 0
13    READ (1,5) A, B, C
 5    FORMAT (3F6.2)
      AVEG = (A+B+C)/3.0
      WRITE (3,6) A, B, C, AVEG
 6    FORMAT (3F6.2, F10.2)
      INT = INT + 1
      IF (INT.NE.5) GO TO 13
      END
      LOADER  RUN
      $EXECUTION
```

Input data;

2.0	3.0	7.0
1.5	2.5	4.4
10.0	20.0	30.0
20.3	30.1	40.5
100.	300.	500.

Output;

2.00	3.00	7.00	4.00
1.50	2.500	4.40	2.80
10.00	20.00	30.00	20.00
20.30	30.10	40.50	30.30
100.00	300.00	500.00	300.00

The Arithmetic IF Statement

The arithmetic IF statement is very similar to the logical IF statement. The only difference is that the arithmetic IF statement contains an arithmetic expression inside the parentheses and the logical IF statement contains a logical expression inside the parentheses. The general form is;

IF (a) n_1, n_2, n_3

where (a) is an arithmetic expression of either type integer or type real, and n is the statement number of an executable statement.

The purpose of an arithmetic IF statement is to instruct the computer to execute one of three specified statements, depending on the value of (a). If the expression (a) is negative, the computer executes the statement n_1. If (a) is equal to zero, the computer executes the statement n_2. If (a) is positive, the computer executes the statement n_3.

Illustration. In the partial program below, if A = 3 and B = 5, the statement 2 is executed. If A = B, the statement 4 is executed, and if A = 2 and B = 1, the expression (A - B) is positive and the statement 6 is executed.

```
      .
      .
      .
      IF (A - B) 2, 4, 6
2     Y = (A - B)**2
4     Y = 0.
6     Y = SQRT (A - B)
      .
      .
      .
```

Sample Program. This program finds the square root of a number and it illustrates the use of an arithmetic IF statement. The computer is instructed to read a number, label it X, and store it. If the number is negative, the additive inverse of the number is found and the square root is computed. If the number is positive, the square root of the number is computed immediately. The computer

is then instructed to write the number and the square root. Checking of the number is done by the IF statement. If (0. - X) is negative, the number is positive and the statement 3 is executed next. If (0. - X) is zero, the number is zero and the statement number 4 is executed next. If (0. - X) is positive, the number is negative and the statement 5 is executed next.

```
      FORTRAN  RUN
      $NO MULTIPLY DIVIDE
      $NO DICTIONARY
      READ (1,11) X
   11 FORMAT (F6.2)
      IF (0. - X) 3, 4, 5
    5 X = -X
      Y = SQRT (X)
      GO TO 7
    4 Y = X
      GO TO 7
    3 Y = SQRT (X)
    7 WRITE (3, 12) X, Y
   12 FORMAT (F6.2, F10.4)
      END
      LOADER  RUN
      $EXECUTION
```

Input data;

```
2.00
```

Output;

2.00	1.4142

The DO Statement

In computer programming, the word loop means a sequence of statements which is used more than once in a particular program. The use of loop is an important aspect of programming and most programs usually contain at least one loop. One of several ways of performing a loop is accomplished by the DO statement. The DO statement is an instruction to execute repeatedly a certain set of statements that follow. One DO statement performs four operations.

1. It designates the sequence of statements which is to be iterated.
2. It defines a variable initially to have some specific value.
3. It increases the variable by a given amount after each execution of the sequence of statements.
4. It tests that value to determine if the required number of instructions has been performed.

The general form is;

$$\text{DO n i} = m_1, m_2, m_3$$

where n designates the number of the last of the sequence of statements to be executed. The i is any nonsubscripted integer variable. The m is either an integer or integer variable. Initially, the state-

ments following the DO statement and up to and including the statement numbered n are executed with the value of i equal to the value of m_1. The loop is repeated with the value of i increased by the value of m_3 each time. The looping process is terminated when the value of i exceeds the value of m_2. At this time the computer executes the next executable statement following the statement numbered n. If the value of m_3 is not specified, the computer will assume it to be 1.

The range of the DO statement is that set of statements that will be executed repeatedly following the DO statement. In the partial program below, the value of I is initially 1. The value of I then increases by one each time the loop is executed and when the loop has been performed ten times, the program comes to an end.

```
      .
      .
      .
    7 DO 5 I = 1, 10, 1
      READ (1,51) P, S
   51 FORMAT (2F8.2)
      RATE = P/S
    5 WRITE (3,52) P, S, RATE
   52 FORMAT (2F8.2, F10.3)
      END
```

Loop — the DO statement and statement 5

Range — READ (1,51) P, S through WRITE (3,52) P, S, RATE

The two programs following illustrate the use of a DO statement. The first program does not make use of any DO statement, but the second one does. Each program reads in fifteen numbers and finds the sum of all fifteen numbers.

In the first program the looping process is accomplished by an arithmetic IF statement and an unconditional GO TO statement. The value of INT is 1 during the first execution of the loop and increased by one before each additional repitition. After the last data card is processed, the value of INT is 16 and (INT - 15) is positive. Thus the computer executes the statement 8.

```
    FORTRAN  RUN
    $NO MULTIPLY DIVIDE
    $NO DICTIONARY
    SUM = 0.
    INT = 0
5   INT = INT + 1
    IF (INT-15) 9,9,8
9   READ (1,1) X
1   FORMAT (F6.2)
    SUM = SUM + X
    GO TO 5
8   WRITE (3,1) SUM
    END
    LOADER  RUN
    $EXECUTION
```

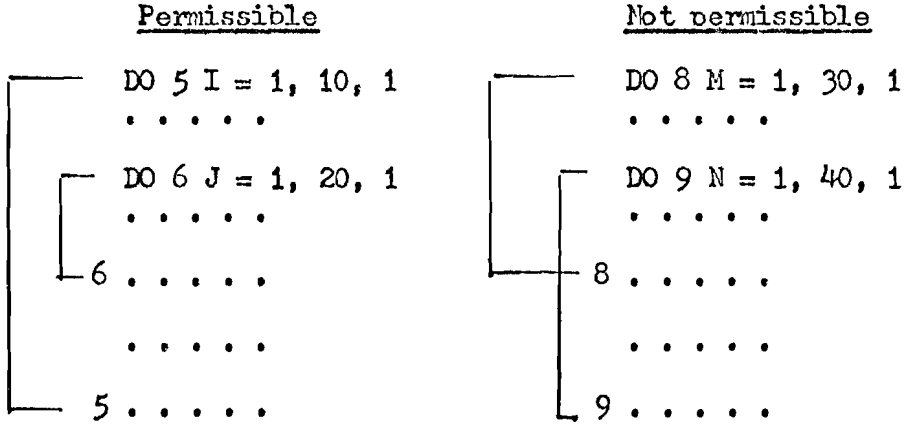
```
    FORTRAN  RUN
    $NO MULTIPLY DIVIDE
    $NO DICTIONARY
    SUM = 0.
    DO 4 I = 1, 15
    READ (1,1) X
4   SUM = SUM + X
    WRITE (3,1) SUM
1   FORMAT (F6.2)
    END
    LOADER  RUN
    $EXECUTION
```

The Nested DO Loop

A DO statement can be contained within another DO statement. The index of each DO statement in a nest must have a different variable. The maximum depth of nesting is twelve. That is, a DO statement can

contain a second DO statement, the second can contain a third, the third can contain a fourth, and so on up to twelve statements.

The flow of the program may be transferred from a statement within a DO loop to a statement outside of a DO loop but the flow of the program cannot be transferred into the range of a DO loop from outside its range.



Any statement that redefines the index is not permitted in the range of a DO loop. The range of a DO loop cannot end with a GO TO type statement or another DO statement. But the range of a DO can end with a logical IF. In this case the control is transferred as follows:

1. If the value of the expression is false, the control returns to the DO statement.
2. If the value of the expression is true, s is executed, and the control returns to the DO statement.

The first and the last statement in the range of a DO statement must be executable. It is recommended that only executable statements be written in a DO loop.

Sample Program. The following program adds two three-by-three matrices and it illustrates the use of a nested DO loop. This program is followed by six data cards. The first data card contains the three numbers of the first row of A matrix. The second data card contains the three numbers of the second row and the third data card contains the three numbers of the third row of A matrix.

The fourth data card contains the three numbers of the first row of B matrix, the fifth data card contains the three numbers of the second row, and the sixth data card contains the three numbers of the third row of B matrix.

The first DO statement reads in the A matrix. When the computer reads in the first data card, the first number is labeled A(1,1), the second number A(1,2), and the third number A(1,3), and stores them. When the second data card is read, the computer labels the first number on this card A(2,1), the second number A(2,2), and the third number A(2,3). When the third data card is read, the computer labels the first number on this card A(3,1), the second number A(3,2), and the third number A(3,3), and stores them.

The second DO statement reads in the B matrix and the procedure is the same as the one described for reading in the A matrix.

The last two DO statements are associated with the addition of the matrices. During the first part of computation J and I are both assigned the value 1. A(1,1) is added to B(1,1), the sum is labeled C(1,1), and the number represented by C(1,1) is printed along with the subscripts I and J. J is now assigned the value 2 and A(1,2) is added to B(1,2), the sum is labeled C(1,2), and it is printed along

with its subscripts. It should be recalled that the value of I is still 1. J is now assigned the value 3 and A(1,3) is added to B(1,3), the sum is labeled C(1,3) and printed along with its subscripts.

The second DO loop is satisfied and the index of the first DO statement I is assigned the value 2 and the second DO loop is repeated. A(2,1) is added to B(2,1) and the sum is labeled C(2,1) and printed. A(2,2) is added to B(2,2) and the sum is labeled C(2,2) and printed. A(2,3) is added to B(2,3) and the sum is labeled C(2,3) and printed.

I is now assigned the value 3, and the second DO loop is repeated for the third time; A(3,1) is added to B(3,1), and so on.

```
      FORTRAN  RUN
      $NO MULTIPLY DIVIDE
      $NO DICTIONARY
      DIMENSION A(3,3), B(3,3), C(3,3)
      DO 5 I = 1, 3
    5 READ (1,1) A(I,1), A(I,2), A(I,3)
    1 FORMAT (3F6.2)
      DO 6 I = 1, 3
    6 READ (1,1) B(I,1), B(I,2), B(I,3)
      DO 7 I = 1, 3
      DO 7 J = 1, 3
      C(I,J) = A(I,J) + B(I,J)
    7 WRITE (3,2) C(I,J), I, J
    2 FORMAT (F8.2, 2I4)
      END
```

LOADER RUN

$EXECUTION

Input data:

3.00	2.00	4.00
1.00	6.00	2.00
7.00	5.00	9.00
3.00	6.00	8.00
5.00	1.00	3.00
3.00	2.00	1.00

Ouput:

6.00	1	1
8.00	1	2
12.00	1	3
6.00	2	1
7.00	2	2
5.00	2	3
10.00	3	1
7.00	3	2
10.00	3	3

The Implied DO Loop

The implied DO loop is very similar to the regular DO loop in that a sequence of statements is executed more than once. However, unlike the regular DO statement, the implied DO loop does not use the DO statement. Instead, the repetition is accomplished by the subscript method.

The following partial program illustrates the use of the implied DO loop in a READ statement. It is desired that an array of numbers consisting of four rows and five columns be read into the computer. M represents the number of rows and N represents the number of columns. The following four statements accomplish this.

```
      READ (1,1) M, N
    1 FORMAT (2I4)
      READ (1,2) ((A(I,K), I=1,N), K=1,M)
    2 FORMAT (F8.2)
```

The first data card contains two integers, M and N, four and five repectively. At first, the subscript I is assigned the value 1 and the K is assigned 1 also. With the value of K fixed, I varies from one to five. The first number is labeled A(1,1), the second A(1,2), and so on. When the numbers of the first row are read in, the K is assigned the value 2 and I varies from 1 to 5. The sixth number is labeled A(2,1), the seventh number A(2,2), and so on. When the value of K is 4, the last row is read in and all numbers are stored in the core storage. The disadvantage of this statement is that each number must be punched on a card individually.

The CONTINUE Statement

The CONTINUE is a dummy statement that does not produce any executable instructions. However, if it is desired to return to the DO statement in such a way that the index will be incremented, but the last intended statement in the DO range is one that is not permissible, such as a GO TO statement, then a CONTINUE is used as the last statement of the DO range.

The partial program below contains the statement 12 which is not to be executed with each repetition of the loop but only when the value of K is 1. The value of K is limited to 1 and 2. The statement 12 is the last intended statement in the DO range but it is to be executed only when the value of K is 1. Since it is not to be executed with each performance of the loop, it cannot be the last statement in the range. When the GO TO statement is executed, if the value of K is 2, the control is transferred to the statement 8 and from there to the DO statement causing the value of I to increment. The CONTINUE statement serves as a very convenient last step of a DO range.

```
 5  DO 8 I = 1, 100, 1
    READ (1,1) P, K
 1  FORMAT (F8.2, I4)
    GO TO (12, 8), K
12  TOT = TOT + P
 8  CONTINUE
```

The STOP Statement

This statement also halts the computer but the student in the

programming course is asked not to use this statement since several programs are processed at the same time and pressing the START key will not start the computer to process the next program.

The END Statement

This statement must be the last statement in every FORTRAN program and it defines the end of a program. This statement informs the compiler that it is the end of the program. The END statement is not executable and a program may contain only one END statement. The general form is;

END

The DIMENSION Statement

This statement provides the compiler with the information necessary to assign and locate storage spaces for arrays of numbers. When the compiler encounters a non-subscripted variable, it assigns a single storage location to that variable and the variable is referred to by its address. When a subscripted variable is used in a program, the size of the array must be designated to the compiler since many numbers are represented by a single variable. The compiler sets aside the right amount of storage space for each subscripted variable. The general form is;

DIMENSION $v_1(k_1)$, $v_2(k_2)$, ...

where v is the variable name and k is composed of one, two, or three integers separated by commas. This number specifies the size of the array.

A single DIMENSION statement can specify the dimension of any

number of arrays. The DIMENSION statement must precede the first appearance of each subscripted variable. The Program B in the Appendix illustrates the use of this statement.

The COMMON Statement

There are at least two occasions in which sharing of the storage may be necessary; (1) the information used in the main program is used again in a subprogram; and (2) the main program uses a certain amount of storage space for a temporary work and a subprogram uses a storage space for a temporary work. With a COMMON statement, both the main program and the subprogram can use the same storage location. Sometimes a program which is too long for a smaller computer to handle can be written shorter with a proper use of COMMON statements. The general form is;

COMMON a, b, c, ...

where a, b, c, ..., are the names of variables assigned to a common storage. Usually they are array names which can be dimensioned.

If one part of a program has the statement COMMON A and a second part of the program or a subroutine has the statement COMMON B, the variables A and B will share the same storage location. If the main program contains the statement COMMON A, B, C, and a subprogram contains the statement COMMON X, Y, Z, then A and X will occupy the same location, B and Y will occupy the same location, and C and Z will occupy the same location.

A COMMON statement may contain a dummy name. That is, if the main program contains the statement,

```
COMMON A, B, C
```

and it is desired that the array R in the subprogram share the same location as the array B. This can be accomplished by writing in the subprogram,

```
COMMON S, R
```

where S is a dummy array name. The arrays B and R will occupy the same location.

The storage area referred by the COMMON statement must be dimensioned and the dimension statement must precede the COMMON statement. However, it is possible to use COMMON statement with dimensions. The first two statements below can be replaced by the third statement.

```
DIMENSION A(100), C(30)
COMMON A, C

COMMON A(100), C(30)
```

It is important to realize that the variables used in the COMMON statement in the main program and the variables used with the COMMON statement in the subprogram must correspond in type and in order. The first statement below belongs to the main program and the second statement belongs to the subprogram. It should be noted that A and C share the common location and they are both of the same type, that is, they are both real variables. J and I share the same location and they are both integer variables.

```
.
.
COMMON A, J, K, R, S
.
```

```
      .
      .
      .
      COMMON C, I, M, X, Y
```

The EQUIVALENCE Statement

The EQUIVALENCE statement, a non-executable statement, is very similar to the COMMON statement in that both are used to conserve storage space. The COMMON statement provides a facility of having a main program and its subprogram reference the same location. The EQUIVALENCE statement provides a means whereby the same location within a single program may be used for the storage of more than one variable or array. The general form is;

```
EQUIVALENCE (a, b, c, ...), (s, t, u, ...), ...
```

In the sample statement below, variables A, D, and F share the same location and P(I) and Q(J) share the same location.

```
EQUIVALENCE (A, D, F), (P(I), Q(J))
```

The EQUIVALENCE statement can be placed anywhere in the program. Once the memory allocation has been made, it cannot be changed. It is improper to write another EQUIVALENCE statement which contradicts the previous allocation. The variables that are made equivalent must be of the same type and none of the dummy arguments may appear in an EQUIVALENCE statement.

The Type Statement

The purpose of a type statement is to specify the type of numbers

o be associated to a variable name. The type statement makes it ossible to use real variables as integer variables and integer variables s real variables. The general forms are;

```
INTEGER a, b, c, ...
REAL a, b, c, ...
EXTERNAL p, q, r, ...
```

here a, b, c, ... are variable names appearing within the program and , q, r, ... are function or subroutine names appearing as actual rguments within the program.

The appearance of a name in the list of a type statement nullifies he predefined type indicated by the first letter of the name. The ppearance of a name in the list of type statement designates the type ermanently in the program and it may not be changed. A name can ppear in only one type statement and the statement must be placed in he program such that it precedes the first use of it.

The first statement below specifies that all the names listed re of integer type and the names COST and BETA assume integer values. n the second statement names K and Max assume real values. The names AT and BEE are names of a FUNCTION or SUBROUTINE subprogram and they re passed as an argument from one program to another.

```
INTEGER COST,  BETA
REAL K, MAX
EXTERNAL CAT, BEE
```

Sample Program. The following program finds the area of a ircle whose radius is N which is an integer variable but the type

statement made it a real variable and it can be multiplied to a real number.

```
      FORTRAN  RUN
      $NO MULTIPLY DIVIDE
      $NO DICTIONARY
      REAL N
      READ (1,5) N
    5 FORMAT (F6.2)
      A = 3.14*N**2
      WRITE (3,5) A
      END
      LOADER  RUN
      $EXECUTION
```

The DATA Statement

This is a non-executable statement and its purpose is to assign initial values to ordinary and subscripted variables. The general form is;

DATA list/d_1, d_2, ..., d_n/, list/d_1, d_2, ..., d_n/, ...

where list contains ordinary or subscripted variables separated by commas and d's are the initial values to be assigned to each of the names in the associated list separated by commas.

Illustrations.

Statement	DATA R/34.5/
Stored value	R = .345 x 10^2

Statement	DATA A, B, K/ 32.7, 0.05, 54/
Stored values	$A = .327 \times 10^{2}$
	$B = .5 \times 10^{-1}$
	$K = 54$

There must be a correspondence between the initial values and the names in the list. The value of d_1 is assigned to the first name in the list and d_2 is assigned to the second name and so on. It should be noted that the variable names appearing in a DATA statement cannot appear in a COMMON statement.

CHAPTER VI

INPUT/OUTPUT STATEMENTS

The input statements handles the transmission of data between the computer and input devices such as the card reader and the output statements handles the transmission of data between the computer and the output devices such as the card punch or printer. The I/O statements fall into one of the following general categories; (1) FORMAT statements; (2) General I/O statements; (3) Manipulative I/O statements; and (4) I/O specification statements.

List Specifications. An I/O list is a series of items that are separated by commas. A single list item can be a subscripted or non-subscripted variable. An I/O list is read from left to right. The data card for the following partial program should have three numbers and the first number on the data card will be labeled as X, the second number as Y, and the third number as P.

```
      .
      .
      .
      READ (1,5) X, Y, P
    5 FORMAT (3F8.2)
      .
      .
      .
```

An I/O list is ordered. The order must be the same as the order in which the numbers appear on the data card. The order must also be maintained when numbers are printed.

The READ or WRITE statement can contain one or more implied DO's. The I/O statement containing parenthese is executed in a manner

similar to the execution of a DO statement. The left parentheses (except subscripting parentheses) are treated as though they were a DO statement. The two statements on the left are equivalent to the three statements on the right.

```
    READ (1,3) (X(I), I = 1,10)          DO 8 I = 1, 10
3 FORMAT (F8.2)                        8 READ (1,3) X(I)
                                       3 FORMAT (F8.2)
```

The FORMAT Statement

The FORMAT statement tells where and in what form the data appears on the card if the statement is used with a READ statement. It also tells where and in what form the data is to be printed if the statement is used with a WRITE statement. The FORMAT statement is non-executable statement which must always be used with a READ or WRITE statement. The general form is;

$$n \text{ FORMAT } (s_1, s_2, \ldots, s_n)$$

where n is the statement number and s is a series of specifications separated by commas. Each FORMAT statement must always be given a statement number and the specifications must be ordered and consistent with the data on the input or output record. There are three types of specifications. In the following list, I, F, and E indicate the type of numbers, w indicates the width of a number or the number of digits in a number, and d indicates the number of digits to the right of the decimal point.

Specification	Type of Number	Example
Iw	Integer	FORMAT (I4)
Fw.d	Real without exponent	FORMAT (F6.2)
Ew.d	Real with exponent	FORMAT (E10.4)

The data card for the partial program below must contain three numbers. The first number, which will be labeled K, must be an integer with the width of four digits. The second number on the data card, which will be labeled S, must be a real number without an exponent with the width of six, of which two are to the right of the decimal point. The third number on the data card, which will be labeled T, must be a real number with an exponent with the width of ten, of which four are to the right of the decimal point.

```
      .
      .
      .
      READ (1,4) K, S, T
    4 FORMAT (I4, F6.2, E10.4)
```

The w or the width of a number can be greater than that required for the actual digits. This is often done to provide spacing of the numbers. For example, the statement, FORMAT (I3, E12.4, F10.4) causes the following line to print.

Number in storage	27	-64.8923	-0.007634
Specification	I3	E12.4	F10.4
Printed line	b27b-0.6489Eb02bbb-0.0076	(b indicates a blank)	

In determining the width w for E-specification seven spaces

must be reserved in addition to the number of digits to the right of the decimal point. The following accounts for the seven spaces; one for the sign, one for the decimal point, one for a possible zero that precedes the decimal point if the absolute value of the number is less than one, one for E, one for the exponent sign, and two for the exponent. Thus $w \geq d + 7$.

In determining the width for F-specificaion, three spaces must be reserved in addition to the number of digits to the right of the decimal point. The sign, the decimal point, and a possible zero that precedes the decimal, each take up a space.

For I-specification only one additional space is necessary for the sign. Caution must be taken in punching integer numbers when I-specification is used. In I-specification the numbers must be right justified. Blanks cannot be used to the right of the number. If a number is read with FORMAT (I5), and the number is 25, 2 must be punched in column four and 5 in column five. If 2 is punched in column three and 5 in column four, the number is read in as 250.

The following are examples regarding the type of output various specification will produce. The number stored in the memory is -345.6.

Specification	Output
F5.0	-345.
F6.1	-345.6
F7.1	b-345.6
E10.2	-34.5E+01

The specification F5.2 is not acceptable since the width of 5 is

not large enough to accomodate the number. If two digits are to be used to the right of the decimal point, the width must be at least 7. The specification E9.4 is also incorrect for this number. The width must be at least 11. The following are examples regarding the type of output various specifications will produce. The number stored in the memory is +22.E+5.

Specification	Output
E8.0	b22.E+05
E10.0	b22000.E+02
E15.6	bbb2.200000E+06

The specification E15.7 will not accomodate the number since the width is not large enough if seven digits are to be allowed to the right of the decimal point. The number in the memory is 15 for the following illustrations.

Specification	Output
I3	b15
I6	bbbb15

The specifications I1 and I2 are incorrect for this number.

Alphameric Fields

The alphameric characters consist of numbers 0 through 9, letters A through Z, and other characters such as $, =, blank, /, -, (,), (.), +, and *. Alphameric values rather than numeric values may be substituted for variables. For example, the variable COST may be replaced by the letter P just as in the case where X may be replaced

by 45.3.

A-Conversion

In order to read in or write out the alphameric data, A-specification must be used. The general form is,

```
FORMAT (nAw)
```

where n indicates the number of repetition and w indicates the width of the field or the number of characters.

If an input record contains one character B and it is desired that a five character variable, DELTA, to be defined as the value B. The necessary statements are as follows;

```
      READ (1,3) DELTA
    3 FORMAT (A1)
```

The following sample program illustrates the use of alphameric data. It reads in the letters A through Z and stores A in XA, B in XB, C in XC, and so on. Then, the computer prints two words, KANSAS and COLLEGE.

```
      FORTRAN  RUN
      $NO MULTIPLY DIVIDE
      $NO DICTIONARY
      READ (1,3) XA, XB, XC, XD, XE, XF, XG, XH, XI
      READ (1,3) XJ, XK, XL, XM, XN, XO, XP, XQ, XR
      READ (1,3) XS, XT, XU, XV, XW, XY, XX, XZ, X-
    3 FORMAT (9A1)
```

```
      WRITE (3,4) XK, XA, XN, XS, XA, XS,
      WRITE (3,5) XC, XO, XL, XL, XE, XG, XE
    4 FORMAT (6A1)
    5 FORMAT (7A1)
      END
      LOADER  RUN
      $EXECUTION
```

Input data:

```
A B C D E F G H I
J K L M N O P Q R
S T U V W Y X Z -
```

Output:

```
K A N S A S
C O L L E G E
```

Since the FORMAT specification of the input is A1, the letters on each input card must be punched without any space between letters and the first letter must be punched in column one.

H-Conversion

The H-specification is used chiefly to print messages and headings in the output. The general form is;

nH

where n represents the number of characters following the H.

Sample Program. This program reads in two numbers, finds the

sum and the difference of the two numbers, and prints the headings and the results.

```
      FORTRAN  RUN
      $NO MULTIPLY DIVIDE
      $NO DICTIONARY
      INTEGER SUM, DIFF
      READ (1,6) M, N
    6 FORMAT (2I5)
      SUM = M + N
      DIFF = M - N
      WRITE (3,21)
      WRITE (3,22)
      WRITE (3, 23) SUM, DIFF
   21 FORMAT (14H ASSIGNMENT #3)
   22 FORMAT (15H JOHN PATTERSON)
   23 FORMAT (7H SUM = , I5, 14H DIFFERENCE = , I5)
      END
      LOADER  RUN
      $EXECUTION
```

Input data:

```
7   4
```

Output:

```
ASSIGNMENT #3
JOHN PATTERSON
SUM =    11 DIFFERENCE =     3
```

It is recommended that the space following the H is left blank, unless the student wishes double spacing in the printing of output records. In that case the numeral 0 is punched in the space immediately following the H. The student must be careful not to punch the letter O instead of the numeral 0. The letter O following the H will cause the printer to dispense many feet of printing paper uselessly.

Blank Fields (X-Conversion)

The general form is;

nX

Used in an input FORMAT statement, this causes the next n characters in the input record to be skipped or ignored regardless of what they are. Used in an output FORMAT statement, this causes n blank characters to be inserted in the output record. This is useful in providing a space between successive output fields.

Sample Program. This program simply prints the two words, PRODUCT and QUOTIENT, and two numbers represented by M and N. M = 123 and N = 321.

```
      FORTRAN  RUN
      $NO MULTIPLY DIVIDE
      $NO DICTIONARY
      WRITE (3,6)
      WRITE (3,7) M, N
    6 FORMAT (8H PRODUCT, 5X, 9H QUOTIENT)
    7 FORMAT (I7, 5X, I8)
```

```
END
LOADER  RUN
$EXECUTION
```

Output:

```
PRODUCTbbbbbQUOTIENT  (line 1)
bbbb123bbbbbbbbbb321  (line 2)
```

Repetition of Field FORMAT

Whenever input or output fields have identical FORMAT specifications, it is not necessary to repeat the FORMAT specification within one FORMAT statement. This is accomplished by placing an integer in front of the E, F, I, or A. The second FORMAT statement is equivalent to the first one.

```
3  FORMAT (F6.2, F6.2, F6.2)
3  FORMAT (3F6.2)
```

Repetition of Groups. It is also possible to repeat a group of specifications by placing an integer in front of the left parenthesis which contains the group of specifications. The following two statements are equivalent.

```
4 FORMAT (I5, F6.2, E14.2, I5, F6.2, E14.2)
4 FORMAT (2(I5, F6.2, E14.2))
```

It is not permissible to have a repetitive group within another repetitive group. The statement below is not permissible.

```
5  FORMAT (2(I5, F6.2, 3(E14.2, I5)))
```

All three statements below are equivalent.

```
6  FORMAT (I3, F6.2, E14.2, E14.2, F6.2, E14.2, E14.2, I5)
6  FORMAT (I3, F6.2, 2E14.2, F6.2, 2E14.2, I5)
6  FORMAT (I3, 2(F6.2, 2E14.2), I5)
```

Scale Factors (P-Conversion)

The purpose of P-specification is to relocate the decimal point in a real number during the process of input or output. A scale factor is an integer constant which indicates the number of places the decimal point is to be moved to the left or right of its original location. The general form is;

nPs

where n is the scale factor and s is the FORMAT specification. A positive integer moves the decimal point to the left while a negative integer moves it to the right. For input, a scale factor may only be used with F specification.

If it is desired that the decimal point in 342.67 is moved two places to the left, the specification required for this is 2PF7.2 and the new number is 3.4267. If the real number 342.67 is to be changed to 3426.7, the appropriate specification is -1PF7.2.

For output a scale factor may only be used with F and E specifications. The explanation given for the input is also appropriate for the output. (See Program D in the Appendix)

A positive scale factor used for output with E-specification increases the base and decreases the exponent. A negative scale factor decreases the base and increases the exponent.

In the illustration below, the origianl number is 0.32457E+03.

FORMAT	Output
1PE12.5	b3.24570E+02
-1PE12.5	b0.03245E+04

Note. Once the scale factor is given in a FORMAT specification, it applies to all E and F specifications following the scale factor within the same FORMAT statement. The following two statements are equivalent.

```
6  FORMAT (1PE12.5, E14.7, F6.2)
6  FORMAT (1PE12.5, 1PE14.7, 1PF6.2)
```

If it is desired that only the first item in a statement be affected by a scale factor, the statement should be written in the following manner.

```
FORMAT (1PE12.5, 0PE14.7, F6.2)
```

Multiple-Record FORMAT Statement

A single FORMAT statement can be used to accommodate more than one input or output record. Separation by (/) indicates the beginning of a new record. For example,

```
3 FORMAT (2F6.2, E14.4/4E12.3)
```

transfers the first, third, fifth, ..., records with the specification 2F6.2, E14.4, and the second, fourth, sixth, ..., records with the specification 4E12.3. (See the Program D in the Appendix)

The use of two consecutive slashes, (//), causes an input record to be read but ignored or an output record to be blank. For example,

```
5 FORMAT (2F6.2//I5//)
```

processes the first, sixth, eleventh, ..., records with the specification 2F6.2, the second, seventh, twelfth, ..., records are blank, the third, eighth, thirteenth, ..., records with the specification I5, the fourth, nineth, fourteenth, ..., records are blank, and the fifth, tenth, fifteenth, ..., records are also blank. It should be noted that there is no specification following the second (//).

In a single multiple-record FORMAT statement, if it is desired that the first two records are unique and all remaining records are to be processed with the same specification, the specification of the remaining records must be defined as repetitive group by enclosing it in parentheses. For example,

```
4 FORMAT (I5/2F6.2/(8F10.2))
```

will process the first record with the specification I5, the second with the spocification 2F6.2, and all remaining records with the specification 8F10.2. The four statements on the left are equivalent to the two statements on the right.

```
3 FORMAT (F6.2, I5)                3 FORMAT (F6.2, I5/F8.2, I4)
4 FORMAT (F8.2, I4)                  WRITE (3,3) P, K, R, M
  WRITE (3,3) P, K
  WRITE (3,4) R, M
```

Carriage Control

The carriage control controls the spacing of output records on the printer. The carriage control character must appear in the space immediately following the H in a FORMAT statement. The three characters used for spacing are as follows.

Character	Result
Blank	Single space printing.
0	Double space printing.
1	Skip to the first of the following page.

Illustrations. The first four statements illustrate the single space printing and the second four statements illustrates the double space printing.

Statement	Output records	
1 FORMAT (1H , F6.2)	Line 1	bbb3.24
WRITE (3,1) A	Line 2	bb23.56
WRITE (3,1) B	Line 3	b 278.54
WRITE (3,1) C		
2 FORMAT (1H0, F6.2)	Line 1	bbb3.24
WRITE (3,2) A	Line 2	
WRITE (3,2) B	Line 3	bb23.56
WRITE (3,2) C	Line 4	
	Line 5	b278.54

Edited Input Data

Edited input data must correspond in order, type, and field width to the field specification in the FORMAT statement and must conform to the following specifications.

1. Plus signs are indicated by a blank or + and minus signs are indicated by a preceding -.

2. Numbers for E and F-specification can contain any number of digits but only the high-order f digits will be retained. The number is truncated to f digits of accuracy. For the IBM 1401, f is 8 unless otherwise specified. The absolute value of the number must be between the limits 10^{-100} and $(1 - 10^{-f}) \times 10^{99}$. Numbers for I-specification must be right justified.

3. Numbers for E-specification need not have four columns devoted to the exponent field. The start of the exponent field must be represented by an E, or if the E is omitted, by a + or -, not a blank. The following expresseions are permissible.

.234E+02, .234E 02, .234E+2, .234+2, .234-44

4. Numbers for E and F-specification need not have the decimal point punched. The computer will supply the necessary decimal point according to the FORMAT specification.

Data	Specification	Result
-08765+2	E12.4	-0.8765+2
05438	E12.3	0.544

The READ Statement

The READ statement is used to transfer input data from an input device to the memory unit of the computer. The general forms are;

```
READ (i,n) list
READ (i)
READ (j'e,n) list
READ (j'e) list
```

i ... is the symbolic unit number of an input device. This number indicates from which one of several input devices the data is to be read. The symbolic name of the card reader at the Data Processing Center is 1. This number must always be an integer or integer variable.

n ... is the statement number of the FORMAT statement by which the data is read.

j ... is an unsigned one digit integer or integer variable which specifies a specific memory space on a disk unit from which the data is to be read.

' ... is a 4-8 punch, equivalent to the @ symbol.

e ... is an unsigned integer, integer variable, or integer expression that refers to a specific record within the memory space on a disk.

list ... is an ordered series of variable names separated by commas.

The READ (i,n) list. This statement tells the computer to read data according to the FORMAT statement n and store the data in the core storage until the list is satisfied.

The READ (i) list. This statement is used to transfer the data from a tape or disk unit to the core storage. The data processed is unedited. This means that the data is processed without any FORMAT specification. Therefore, the data will be stored in the core storage in the same form as it is stored in the disk or tape.

The READ (j'e,n) list. This statement is used to transfer the edited data from a tape or disk unit. Each record is read in order and in accordance with the FORMAT statement n until all the data in the input list have been read and stored.

The READ (j'e) list. This statement is used to transfer data from a tape or disk unit without any FORMAT specification. With this statement, the programmer can specify the record to be read by substituting e with the number which corresponds to the position of the data.

Every number read in as an input data must be represented by a variable. If an integer 274, punched in columns 3, 4, and 5 on a card, is to be read in as an input data, the two statemens below will accomplish this.

```
      READ (1,5) M
    5 FORMAT (I5)
```

If two numbers, 35 and 46, are to be read in as input data and they are both punched on a single card as four digit numbers, each number must be represented by a variable. The necessary statements to accomplish this are as follows.

```
      READ (1,22) K, L
   22 FORMAT (2I4)
```

If two numbers are punched on separate cards in the first four columns, the following statements are needed.

```
      READ (1,10) M
      READ (1,10) N
   10 FORMAT (I4)
```

The first number is represented by M and the second number is represented by N. If many numbers are to be read in it is better to represent the numbers by a subscripted variable.

For the information concerning the READ statement associated with the disk, the student should read Chapter VII.

Sample Program. This program adds K numbers and illustrates the use of READ statement. The computer is instructed to read K which is the number of numbers to be added. Then the computer is instructed to read the first number, label it AMES and add to TOT which is 0 at this time. The second number is read, labeled AMES, added to TOT, and the sum is labeled TOT. Third number is read, labeled AMES, added to the previous sum TOT and the new sum is labeled TOT. This process continues until K number of cards are read. The variable IN counts the number of data cards processed.

```
      FORTRAN  RUN
      $NO MULTIPLY DIVIDE
      $NO DICTIONARY
      READ (1,18) K
   18 FORMAT (I12)
      TOT = 0.
```

```
      IN = 1
   21 READ (1,16) AMES
   16 FORMAT (F8.2)
      TOT = TOT + AMES
      IF (IN.EQ.K) GO TO 40
      IN = IN + 1
      GO TO 21
   40 WRITE (3,16) TOT
      END
      LOADER  RUN
      $EXECUTION
```

The WRITE Statement

The WRITE statement is used to transfer data from the core storage to output devices. The general forms are;

```
WRITE (i,n) list
WRITE (i) list
WRITE (j'e,n) list
WRITE ( j'e) list
```

i ... is the symbolic name of an output device. This symbol indicates from which one of several output units the data are to be processed. It must be one digit integer or an integer variable. The symbolic name of the printer at the Data Processing Center is 3, and the symbolic name of the card puncher is 2.

n ... is the statement number of a FORMAT statement by which

the data is written.

j ... is an unsigned one digit integer or an integer variable which specifies a special memory space on a tape or disk where the data is stored.

' ... is a 4-8 punch equivalent to the @ symbol.

e ... is an unsigned integer, integer variable, or integer expression that refers to a specific record within the memory space on a tape or disk.

list is an ordered series of variable names to be written separated by commas.

The WRITE (i,n) list. The purpose of this statement is to tell the computer to write the data through the output device indicated by i and according to the FORMAT statement n.

The WRITE (i) list. This statement is used to transfer the unedited data from the core storage to a tape or disk unit. The data will be written in the same form as it is stored in the core storage.

The WRITE (j'e,n) list. This statement transfers the data from the core storage to the memory space whose symbolic name is designated by j. The data is transferred according to the FORMAT statement n.

The WRITE (j'e) list. This statement transfers the data from the core storage to a tape or disk unit in the same form as it is stored in the core storage. The programmer can specify the record to be written to be placed at a certain position in the memory space by substituting the position number for e.

Whenever a number stored in the core storage is to be printed in

an output the number must be represented by a variable in the list of a WRITE statement. Every number stored in the core storage must have a variable name. The following program reads in two numbers, finds the sum, and stores the result in a location called SUM and the sum is 35. Then SUM is the variable which represents the number 35. The WRITE statement must have SUM in the list. This is illustrated by the following partial program.

```
      READ (1,11) A, B
11    FORMAT (2F8.2)
      SUM = A + B
      WRITE (3,12) SUM
12    FORMAT (F10.2)
```

The information concerning the WRITE statement associated with the disk can be found in Chapter VII.

The Find Statement

This statement is used to save computing time when a tape or disk is used. This causes the access arm of a disk unit to move to the next record to be processed while computing is going on. Therefore, the greater the distance is between the last record and the next record, the more time this statement will save. The general form is;

```
FIND (j'e)
```

where j is an unsigned one digit integer or an integer variable which refers to the memory space on the disk. This j corresponds to the j in READ or WRITE statements. The letter e is an unsigned integer,

integer variable, or, integer expression which refers to the position of the record where the access arm is to be placed. This e corresponds with the e in the READ or WRITE statement. This statement is used in the Program C in the Appendix.

CHAPTER VII

STORING IN DISK

Information can be transferred from the core storage to the disk storage or from the disk storage to the core storage. When a program processes a large amount of data, the programmer may be forced to store the data in the disk storage in order to conserve the core storage for storing and processing instructions.

In order to store information in the disk storage the programmer must inform the compiler; (1) what part of the disk is to be used and what name is to be assigned to this part; and (2) how many numbers are to be stored in the reserved area and how long each number is.

The ASGN card accomplishes (1) and the DEFINE FILE statement accomplishes (2).

ASGN Card.

This card instructs the computer to reserve a certain area on the disk for storing information and assigns a name to this area. The name assigned to the reserved area on the disk must be one of the following three names; WORK4, WORK5, or WORK6. The programmer may reserve three distinct areas on the disk for storing information. The storage areas, named WORK1, WORK2, and WORK3, are used by the compiler and they are not available to the student. The following is an example of an ASGN card.

```
WORK6 ASGN 1311 UNIT 2, START 001500, END 002000
```

The WORK6 is punched beginning at the column 6 and ASGN and the remaining words and numbers are punched beginning at the column 16 with a single space between the words and the numbers. The address number must have six digits.

Each functional component of the computer has a name. For instance, the card reader is given the name INPUT and the printer is given the name LIST. For the sake of simplicity each of these components is also given a numerical name. These numerical names can be considered as the address of the component.

When the programmer reserves an area on the disk and gives it the name WORK6, the corresponding numerical name for this is 9. The number 8 is associated with the WORK5 and 7 is associated with the WORK4, as 1 is a numerical name given to the card reader and 2 is the numerical name given to the card punch.

The DEFINE FILE Statement

This statement informs the computer the maximum number of records which are to be stored in the area reserved with the ASGN card and how many characters each record may contain. The general form is;

DEFINE FILE $j_1(m_1, l_1, f_1, v_1), j_2(m_2, l_2, f_2, v_2), \ldots$

The j is an integer which is the numerical name of the area reserved on the disk. The reader will recall that 9 is assigned to the WORK6, 8 to the WORK5, and 7 to the WORK4. This number appears first inside the parentheses of READ or WRITE statements. For example, when the computer encounters READ (9'I) it goes to the area on the disk

whose name is 9.

The m is an integer which indicates the maximum number of records which are to be stored in the area on the disk.

The l is an integer. If the numbers transferred from the core storage to the disk according to a FORMAT specification, the l represents the maximum number of characters in each record. If the numbers are transferred to the disk without any FORMAT specification, that is, the numbers are transferred to the disk as they appear in the core storage, the l represents the maximum number of numbers each record contains.

The f is either E or U. If the numbers are stored according to a FORMAT specification E is used. If the numbers are stored without any FORMAT specification U is used.

The v is an integer variable name. The student may use any integer variable. The DEFINE FILE statement informs the computer that this variable is used as the position counter in the READ or WRITE statement. The number of position from which the number is to be read or in which the number is to be written is substituted for this variable before the appearance each READ or WRITE statement. Therefore, this variable must appear inside the parentheses of each READ or WRITE statement. If 1 is assigned to this variable before the READ statement is executed, the record in the first position is read. If the numerical value of this variable is 7, the seventh record is read.

Writing the data on the disk is accomplished with one of the following two WRITE statements.

WRITE (j' e, n) list

WRITE (j' e) list

where j is the same as the j and e is the same as the v in the DEFINE FILE statement. The n is the FORMAT statement number.

Reading the data from the disk and storing it in the core storage is accomplished with one of the following two READ statements.

READ (j' e, n) list

READ (j' e) list

where j is the same as the j and e is the same as the v in the DEFINE FILE statement. The n is the FORMAT statement number.

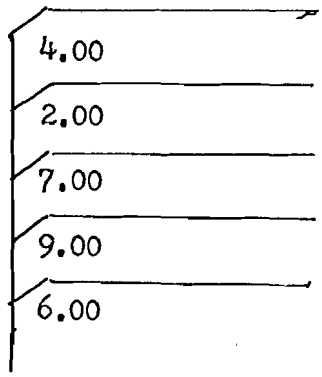
Sample Program. The following program reads in five numbers, one at a time, stores it in the core storage, and transfers it on to the disk. The ASGN card indicates to the computer that the memory space to be reserved on the disk is to be called WORK6 and it begins at the address 1500 and ends at 1510 on the disk unit 2. The DEFINE FILE statement indicates to the computer that the maximum number of records to be stored in the memory space whose name is WORK6 and the numerical name is 9 is 10, each record is 20 characters long at most, the numbers are stored according to a FORMAT specification, and finally, the position of the records stored in the memory space is to designated by the value of INT. The first number is stored in the first position in the memory space which has been reserved since the value of INT is 1, I=1. The second number is stored in the second position since the value of INT is 2, and so on.

After all five numbers are stored on the disk, the third number

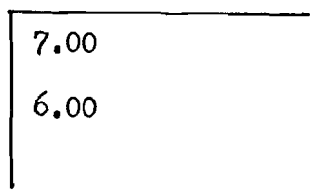
is transferred back into the core and printed and the fifth number is also transferred from the disk to the core and printed. Ten 20-digit records will be stored in ten spaces starting from 1500 and ending at 1510.

```
    WORK6 ASGN 1311 UNIT 2, START 001500, END 001510
    FORTRAN  RUN
    $NO MULTIPLY DIVIDE
    $NO DICTIONARY
    DEFINE FILE 9(10,20,E,INT)
    DO 5 I = 1,5
    READ (1,1) R
1   FORMAT (F6.2)
    INT = I
5   WRITE (9' INT, 2)
2   FORMAT (F8.2)
    INT = 3
    READ (9'INT, 3) S
    WRITE (3,3) S
3   FORMAT (F8.2)
    INT = 5
    READ (9'INT, 3) P
    WRITE (3,3) P
    END
    LOADER  RUN
    $EXECUTION
```

Input data:



Output:



CHAPTER VIII

SUBPROGRAMS

A programmer may wish to use the same sequence of instructions at various points in a program. It would be a waste of time and computer storage space if the set of instructions were written whenever they are needed. It is, however, possible to write the sequence of instructions only once and refer back to it whenever necessary. This sequence of instructions is called a subprogram. The program which utilizes a subprogram is called a calling program.

Three types of subprograms will be discussed in this chapter; (1) library functions; (2) FUNCTION subprograms; and (3) SUBROUTINE subprograms.

In writing a subprogram the student must be aware of the following; (1) the main program must contain the name of the subprogram; (2) some provision must be made for passing data to the subprogram; (3) some provision must be made for receiving values returned from the subprogram back to the calling program, and (4) some provision must be made for returning from the subprogram to the calling program.

The Library Function

If a programmer wishes to find the square root of a number, it is not necessary for him to write a sequence of instructions which computes the square root. A program necessary to compute the square root has been written by another programmer and permanently stored in the computer. The computation of a square root is only one of several functions which are available with IBM 1401. The collection of these

functions is called "Library Functions" and the list of these functions is on page 29 of the IBM 1401 manual.

If a programmer wishes to compute the square root of a number which is represented by a variable X, he would use the first expression below.

```
SQRT (X)
ALOG (2.0)
```

The second expression will compute the natural logarithm of 2.0. Note that some of the functions listed on page 29 will accept only certain type of data. For example, if the square root of 2 is to be computed, it must be written as 2.0. In other words, the function SQRT will not compute the square root of integer constants or integer variables.

The FUNCTION Statement

The purpose of a function subprogram is to receive data from the main program (or calling program), perform operations, obtain a single value equivalent to the value of the function, and return that value to the main program. The function subprogram is a separate and complete program in itself, but it is used by other programs. The general forms are;

FUNCTION name $(a_1, a_2, \ldots, a_n)$
REAL FUNCTION name $(a_1, a_2, \ldots, a_n)$
INTEGER FUNCTION name $(a_1, a_2, \ldots, a_n)$

where name is the name of the function which is written just like the ordinary variable name. It consists of one or more letters and numbers

up to six characters. The first character must be a letter and it must be chosen carefully since it determines the type of values to be processed. The a represents one or more variables names, array names, or simply dummy names.

The REAL FUNCTION statement means that all the variables inside the parentheses are treated as real variables regardless of what they are and the INTEGER FUNCTION statement means that all the variables are treated as integer variables regardless of what they are.

The FUNCTION statement must be the first statement in a FUNCTION subprogram. Following the FUNCTION statement there must be at least one statement which defines the function name, at least one RETURN statement, and one END statement. The RETURN statement is necessary since the value obtained for the function is passed back to the main program when this statement is executed. The END statement is necessary since this statement notifies the compiler that the final statement in the subprogram has been translated.

It is very important to remember that the dummy variables used in the FUNCTION statement must correspond in types to the dummy variables used in the function in the main program. Therefore, the dummy variables must be placed in proper order. As an example, two statements are shown below. The first is the calling statement in the main program and the second is the FUNCTION statement in the subprogram. The arrows indicate the correspondence.

```
SUM =  CALC  (K, A, B, I, X, Y, L)
              ↘  ↘  ↘  ↘  ↘  ↘  ↘
FUNCTION CALC (J, P, R, N, C, D, M)
```

A FUNCTION subprogram cannot contain another FUNCTION subprogram within the program. Also it may not contain a SUBROUTINE statement.

Since a FUNCTION subprogram is a separate and complete program in itself, it is necessary that each FUNCTION subprogram contains the three beginning control cards and the two ending control cards as shown below.

Beginning control cards	Ending control cards
FORTRAN RUN	LOADER RUN
$NO MULTIPLY DIVIDE	$EXECUTION
$NO DICTIONARY	

Sample Program. The program below computes the sum of the squares of a set of numbers using a FUNCTION subprogram. It should be noted that the statement number 7 appears twice in this program, once in the main program and once in the subprogram. This is permissible since the subprogram is a separate and complete program in itself.

```
      FORTRAN  RUN
      $NO MULTIPLY DIVIDE
      $NO DICTIONARY
      DIMENSION A(100)
      READ (1, 3) K
    3 FORMAT (I4)
      DO 7 I = 1, K
    7 READ (1,2) A(I)
    2 FORMAT (E14.2)
      TOTSUM = PARSUM (N,A)
```

```
      WRITE (3,2) TOTSUM
      END
      LOADER  RUN
      $NO EXECUTION
C . . . . . BEGINNING OF THE SUBPROGRAM
      FORTRAN  RUM
      $NO MULTIPLY DIVIDE
      $NO DICTIONARY
      FUNCTION PARSUM (M,C)
      DIMENSION C(100)
      DO 7 I = 1, m
    7 PARSUM = PARSUM + C(I)**2
      RETURN
      END
      LOADER  RUN
      $EXECUTION
```

The SUBROUTINE Statement

This statement accomplishes what the FUNCTION statement does in the previous section but the difference between these two statements is that the FUNCTION subprogram must receive data from the main program but the SUBROUTINE subprogram may or may not receive data from the main program. The SUBROUTINE subprogram can contain its own data.

A SUBROUTINE subprogram is called only by a CALL statement. When the subprogram is completed, the next executable statement following the

CALL statement is executed. Like the FUNCTION subprogram, the SUBROUTINE subprogram is a separate and complete program in itself. It may be translated separately, but ordinarily it is translated following the main program. The general form of the first statement in the SUBROUTINE subprogram is;

SUBROUTINE name $(a_1, a_2, \ldots, a_n)$

where name is an ordinary variable name which is substituted by the programmer. It may consist of one or more letters and numbers up to six characters, but the first character must be a letter. The first letter can be any letter since the type of the name has no significance in a SUBROUTINE statement. The a represents variables separated by commas.

A SUBROUTINE subprogram may not contain a FUNCTION statement or another SUBROUTINE statement within the subprogram. It must contain at least one RETURN statement and one END statement.

Since a SUBROUTINE subprogram is a separate and complete program in itself, it is necessary that each subprogram contain the three beginning control cards and the two ending control cards. Each program must also contain the $NO EXECUTION card since the execution of the program is not desired until the complete program is loaded. The $EXECUTION card, however, must be placed at the end of the last program.

It should be noted that if the variables are listed in the CALL statement and the SUBROUTINE statement, they must correspond in order and in type. Also each statement must contain exactly the same number of variables. The arrows in the following two expressions indicate the

correspondence between variables.

```
CALL  CALC    (K, A, TOT, J, C)
                ↘  ↘   ↘   ↘  ↘
SUBROUTINE CALC (M, S, SUM, N, X)
```

A complete sample program illustrating the use of a SUBROUTINE subprogram is included in the Appendix.

The RETURN Statement

This statement is primarily used in a subprogram. At this point the program returns to the main program. This statement also returns the data obtained in the subprogram back to the main program. The general form is;

RETURN

The CALL Statement

This statement is used only in conjunction with the SUBROUTINE statement. It links the main program with the SUBROUTINE subprogram. The general form is;

CALL name (a_1, a_2, ..., a_n)

where name is the name of the SUBROUTINE subprogram and a represents the variable names or array names which are passed on to the SUBROUTINE subprogram. Each of these names must have been assigned values before they are passed on to the subprogram.

CHAPTER IX

CONTROL CARDS

The control cards are used to give instructions to the computer. These instructions must be punched in the columns designated and must be placed correctly in the program. The student will normally place at the front of any program the following three control cards.

```
FORTRAN  RUN
$NO MULTIPLY DIVIDE
$NO DICTIONARY
```

Each program must also contain the following two control cards at the end of the program. These cards must be placed between the program and the data cards.

```
LOADER  RUN
$EXECUTION
```

FORTRAN RUN

This card is a required control card. This card activates the FORTRAN compiler and the computer starts translating the source statements into the machine language. The card must be punched as follows. The number above the word indicates the column number where the first letter of the word must be punched.

Columns	6	16
Contents	FORTRAN	RUN

LOADER RUN

The FORTRAN RUN card causes the compiler to translate the source program, and the LOADER RUN card causes the translation to be stored in the core storage.

Columns	6	16
Contents	LOADER	RUN

$EXECUTION

The execution of the program begins with this card.

Columns	1
Contents	$EXECUTION

$NO EXECUTION

If the execution of the program is not desired, this card must be placed following the program. This occurs when a FUNCTION subprogram or a SUBROUTINE subprogram is used in the program. This card must be present following each program except the last program.

Columns	1	5
Contents	$NO	EXECUTION

$REAL SIZE and $INTEGER SIZE

These cards are used to designate to the computer the maximum number of digits the data will have.

Columns	1	6
Contents	$REAL SIZE = nn	
Columns	1	10
Contents	$INTEGER SIZE = nn	

$NO MULTIPLY DIVIDE

The computer at the Data Processing Center does not have a unit which performs multiplication and division. A subprogram which performs these operations is contained in the FORTRAN compiler and this control card is necessary to automatically utilize the subprogram.

Columns	1	5	14
Contents	$NO	MULTIPLY	DIVIDE

$NO DICTIONARY

Without this card, the computer will print the Name Dictionary and the Sequence Number Dictionary. The Name Dictionary contains the addresses of all the variables used in a program and the Sequence Number Dictionary contains the address of every machine language statement. The student is required to insert this card in every program.

Columns	1	5
Contents	$NO	DICTIONARY

$NO LIST

Unless this card is included in the program, the computer will print a list of the source program. This card is normally not included in the program.

Columns	1	5
Contents	$NO	LIST

CHAPTER X

CHECKING THE SOURCE PROGRAM

Following are some of the common mistakes made by a beginning programmer.

Spelling. Correct spelling is mandatory in writing the FORTRAN statement such as SUBROUTINE, FUNCTION, FORMAT, etc. Variable names can be spelled any way the programmer wishes but the spelling must be consistent through out the program.

Parentheses. The number of left parentheses must coincide with the number of right parentheses. A mistake is often made when one or more parentheses are used within another pair. For example, the statement below is incorrect; it needs another left parenthesis.

```
A = ((((C+B)*C+D)*D+E)*E+F)*F+G)
```

Commas. The student must place the comma whenever necessary. If more than one variable are used, they must be separated by a comma. In a FORMAT statement each specification must be separated by a comma.

Arithmetic expressions. The student must not mix real numbers and integer numbers. Often the student writes a real number without the decimal point.

Control Cards. The words and symbols on control cards such as FORTRAN RUN, LOADER RUN, $NO MULTIPLY DIVIDE, and others must be punched in the designated column.

Statement Numbers. If a statement is referred to by another statement, it must have a statement number. Two different statement

cannot have the same statement number.

One of the best ways to check the validity of the source program is to "play computer." Using the data to be processed the student goes through each statement step by step, performing all computations by hand. This process should be repeated with the data which are the extremes.

CHAPTER XI

CONCLUSION

The purpose of this paper as stated in the introduction was to discuss the topics in FORTRAN programming which are essential to the student in the Mathematical Programming course at the Kansas State Teachers College. As old machines are constantly replaced by new machines, the student should find out exactly what kind of machines are available at the Data Processing Center when he enrolls in the course.

A complete sample program and a detailed explanation of each statement of the program is provided in one of the introductory chapters of this paper so that the reader may have a better understanding of the other sample programs that follow. Many of the sample programs are complete with the necessary control statements and they are ready to be processed if the reader so desires. Many of the sample programs also contain some sample data and the corresponding output to make the illustration complete.

Even though computer programming is a relatively new field, many books have been written in this field and they are readily available. It is hoped that this paper will help to fill some gaps and make a small contribution to the field of computer programming.

BIBLIOGRAPHY

BIBLIOGRAPHY

Anderson, Decima M. Computer Programming FORTRAN IV. New York: Appleton-Century-Crofts, 1966.

Golden, James T. FORTRAN IV Programming and Computing. Englewood Cliffs: Prentice-Hall, Inc., 1965.

IBM Corporation. FORTRAN IV Language Specifications, Program Specifications, and Operating Procedures. IBM 1401, 1440, and 1460. Systems Reference Library, File No. GENL-25, Form C24-3322-3, 1966.

McCracken, Daniel D. A Guide to FORTRAN Programming. New York: John Wiley & Sons, Inc., 1961.

APPENDIX

PROGRAM A

This program solves a quadratic equation. It also illustrates the use of the FUNCTION and SUBROUTINE subprograms and the Library Function.

```
                              FORTRAN   RUN
FORTRAN COMPILATION  VER 2 MOD 2
$NO MULTIPLY DIVIDE
$NO DICTIONARY
      C
      C ..... SOLVING QUADRATIC EQUATIONS
      C             FUNCTION AND SUBROUTINE SUBPROGRAMS
      C
001           READ (1,52) NUMB4
002        52 FORMAT (I4)
003           IN=1
004         9 READ (1,53) AQ, BQ, CQ
005        53 FORMAT (3F12.5)
006           CALL QUADZ (AQ, BQ, CQ, RQ1, RQ2, KMPXZ)
007           WRITE (3,63)
008        63  FORMAT (3X, 33H QUADRATIC EQUATION VALUES
              3          , 14X, 25H .....ROOTS                )
009           WRITE (3,65)
010        65  FORMAT (8X, 2H A,8X,2H B,8X,2H C, 8X, 8H COMPLEX,
              A8X, 3H R1, 10X, 3H R2)
011           IF (KMPXZ.EQ.0) GO TO 75
012           RADZ=BQ**2-4.*AQ*CQ
013           DENOZ=(2.*AQ)
014           WRITE (3,66) AQ, BQ, CQ, RQ1, RADZ, DENOZ
015        66  FORMAT (1HO, 3F10.3,10X, 4H YES, 5X, F10.3, 3X,
              718H + AND - SQ ROOT (, F8.2,3H )/, F6.2)
016           GO TO 70
017        75 WRITE (3,67) AQ, BQ, CQ, RQ1, RQ2
018        67 FORMAT (1HO, 3F11.3, 12X, 2HNO, 7X, F11.3, 3X, F11.3)
019        70 CONTINUE
020           IF (IN.EQ.NUMB4) GO TO 77
021           IN=IN+1
022           GO TO 9
023        77 CONTINUE
024           END
```

*** $NO EXECUTION

```
                              FORTRAN   RUN
FORTRAN COMPILATION  VER 2 MOD 2
$NO MULTIPLY DIVIDE
$NO DICTIONARY
001         SUBROUTINE QUADZ (AQ, BQ, CQ, RQ1, RQ2, KMPXZ)
002         RQ1=RUUT1 (AQ, BQ, CQ, KMPXZ)
003         IF (KMPXZ.EQ.1) GO TO 90
004         RQ2=RUUT2 (AQ, BQ, CQ)
005      90 RETURN
006         END

                *** $NO EXECUTION


                              FORTRAN   RUN
FORTRAN COMPILATION  VER 2 MOD 2
$NO MUTIPLY DIVIDE
$NO DICTIONARY
001         FUNCTION RUUT1 (AQ, BQ, CQ, KMPXZ)
002         RADZ=BQ**2-4.*AQ*CQ
003         IF (RADZ.LT.0.0) GO TO 25
004         RUUT1=(-BQ+SQRT(RADZ))/(2.*AQ)
005         KMPXZ=0
006         RETURN
007      25 RUUT1=-BQ/(2.*AQ)
008         KMPXZ=1
009         RETURN
010         END

                *** $NO EXECUTION


                              FORTRAN   RUN
FORTRAN COMPILATION  VER 2 MOD 2
$NO MULTIPLY DIVIDE
$NO DICTIONARY
001         FUNCTION RUUT2 (AQ, BQ, CQ)
002         RUUT2=(-BQ-SQRT(BQ**2-4.*AQ*CQ))/(2.*AQ)
003         RETURN
004         END


$EXECUTION
```

```
QUADRATIC EQUATION VALUES                                  .....ROOTS
     A          B          C          COMPLEX          R1              R2

   1.000     -7.000     12.000           NO                4.000           3.000
QUADRATIC EQUATION VALUES                                  .....ROOTS
     A          B          C          COMPLEX          R1              R2

   1.000     -6.000      5.000           NO                5.000           1.000
QUADRATIC EQUATION VALUES                                  .....ROOTS
     A          B          C          COMPLEX          R1              R2

   1.000     -2.000     -8.000           NO                4.000          -2.000
QUADRATIC EQUATION VALUES                                  .....ROOTS
     A          B          C          COMPLEX          R1              R2

   1.000     -2.000     10.000           YES               1.000     + AND - SQ ROOT (   -36.00 )/   2.00
```

PROGRAM B

This program reads in a set of test scores, finds the average and the standard deviation, and sorts the scores. It illustrates the use of the DIMENSION statement, the implied DO loop, and the IF statement.

```
                              FORTRAN   RUN
FORTRAN COMPILATION  VER 2 MOD 2
$NO MULTIPLY DIVIDE
$NO DICTIONARY
      C
      C ............. MOK  TOKKO
      C
      C.................STORING EDITED DATA IN THE DISK
      C
001           DIMENSION A(20), D(20)
002           DEFINE FILE 9(50,20,E,INDEX7)
003           READ (1,11) M
004        11 FORMAT (I4)
005           READ (1,12) (A(I), I=1,M)
006        12 FORMAT (F10.3)
007           WRITE (3,53)
008        53 FORMAT (10X, 13H THE INPUT IS)
      C......PRINT THE INPUT DATA
009           WRITE (3,55) (A(I), I=1,M)
010        55 FORMAT (F10.2)
      C...... STORE THE EDITED DATA INTO THE DISK
011           WRITE (9'I,22) (A(I),I=1,M)
012        22 FORMAT (E12.4)
013           DO 14 I=1,M
014        14 A(I)=0.
015           READ (9'I,22) (A(I),I=1,M)
016           CONST=M
017           SUM=0.
018           DO 31 I=1,M
019           SUM=SUM+A(I)
020        31 CONTINUE
      C....... COMPUTATION OF MEAN AND THE STANDARD DEVIATION
021           AVER=SUM/CONST
022           TOT=0.
023           DO 32 I=1,M
024        32 D(I)=(A(I)-AVER)**2
025           DO 33 I=1,M
026        33 TOT=TOT+D(I)
027           STDEV=SQRT(TOT/CONST)
028           WRITE (3,51) AVER, STDEV
029        51 FORMAT (15H THE MEAN IS .., F8.2, 10X,
             829H THE STANDARD DEVIATION IS..., F8.2)
      C.........SORTING PROCESS
030           LIMIT=M-1
031         5 INT=1
032           DO 88 I=1,LIMIT
033           IF (A(I+1).LE.A(I)) GO TO 88
034           TEMP=A(I+1)
035           A(I+1)=A(I)
036           A(I)=TEMP
037           INT=I
038        88 CONTINUE
```

```
039         IF (INT.EQ.1) GO TO 77
040         LIMIT=INT-1
041         GO TO 5
042      77 CONTINUE
043         WRITE (3,37)
044      37 FORMAT (3X, 17H THE SORTED SCORE)
045         DO 78 I=1,M
046      78 WRITE (3,79) A(I)
047      79 FORMAT (F10.2)
048         END
```

```
LOADER     RUN

$EXECUTION
```

```
        THE INPUT IS
    98.00
    93.00
    95.00
    78.00
    80.00
    82.00
    85.00
    88.00
    92.00
    90.00
    72.00
    69.00
    49.00
    52.00
    57.00
    75.00
    62.00
    41.00
    43.00
    75.00
THE MEAN IS ..    73.80          THE STANDARD DEVIATION IS...    17.34
   THE SORTED SCORE
    98.00
    95.00
    93.00
    92.00
    90.00
    88.00
    85.00
    82.00
    80.00
    78.00
    75.00
    75.00
    72.00
    69.00
    62.00
    57.00
    52.00
    49.00
    43.00
    41.00
```

PROGRAM C

Whenever a large number of data is to be processed, the data can be stored in the disk storage. In order to store the data in the disk, the data must be stored in the core storage first. If all the data is read into the core before transferring to the disk, a large amount of core storage is used. If a number is read into the core and stored in the disk one at a time or, in case of a matrix, one row at a time, a considerable amount of core storage space is saved.

This program is designed to read in two M by N and N by K matrices one number at a time, store it in the disk, and find the product of the two matrices. The uniqueness of this program is that the elements of the two matrices and the elements of the product matrix are stored linearly. That is, after the first row of the first matrix is stored, the second row is stored immediately following the first row, and the third row immediately following the second row, and so on. After the first matrix is stored, the first row of the second matrix is stored immediately following the last row of the first matrix. Then the second row is stored following the first row, and so on.

The following diagram illustrates the way the numbers are stored in the disk. The elements of the first matrix are represented by a_{mn} and that of the second matrix by b_{nk} and that of the product matrix by c_{mk}.

POSITION NO.	1	2	3	....	n	n+1	n+2	n+3	....	n(m-1)	n(m-1)+1	n(m-1)+2	....
STORED NUMBER	a_{11}	a_{12}	a_{13}	....	a_{1n}	a_{21}	a_{22}	a_{23}	....	a_{m1}	a_{m2}	a_{m3}	....

m·n	m·n+1	m·n+2	m·n+3	....	m·n+K	m·n+K+1	m·n+K+2	m·n+K+3	....	m·n+2K	....
a_{mn}	b_{11}	b_{12}	b_{13}	....	b_{1K}	b_{21}	b_{22}	b_{23}	....	b_{2K}	....

m·n+K(n-1)+1	m·n+K(n-1)+2	m·n+K(n-1)+3	....	m·n+K·n	m·n+K·n+1	m·n+K·n+2		m·n+Kn+m·K
b_{n1}	b_{n2}	b_{n3}	....	b_{nK}	C_{11}	C_{12}		C_{mK}

FIGURE 1

DIAGRAM ILLUSTRATING HOW THE ELEMENTS OF THREE MATRICES ARE STORED IN THE DISK UNIT

```
                          FORTRAN   RUN
FORTRAN COMPILATION  VER 2 MOD 2
$NO MULTIPLY DIVIDE
$NO DICTIONARY
      C ...........MOK TOKKO
      C
      C ................. MULTIPLICATION OF TWO MATRICES
      C                    THE ELEMENTS OF THE MATRICES ARE READ IN ONE
      C                    AT A TIME AND STORED ON THE DISK IN ONE ROW
      C
001           DEFINE FILE 9(1000, 20, U, INDEX9)
002        11 FORMAT (3I4)
003        26 FORMAT (2I4, F10.2)
004        12 FORMAT ( F10.2)
005           READ (1,11) M, N, K
006           WRITE (3,41)
007        41 FORMAT (16HOTHE MATRIX A IS)
008           DO 100 I=1,M
009           DO 100 J=1,N
010           READ (1,12) X
      C ................. STORING OF A MATRIX ON THE DISK
011           IND=(I-1)*N+J
012           FIND (9'IND)
013           WRITE (9'IND) X
014       100 WRITE (3,26) I, J, X
015           WRITE (3,38)
016        38 FORMAT (16HOTHE MATRIX B IS)
017           DO 101 I=1,N
018           DO 101 J=1,K
019           READ (1,12) Y
      C ................. STORING OF B MATRIX ON THE DISK
020           IND =M*N+(I-1)*K+J
021           FIND (9'IND)
022           WRITE (9'IND) Y
023       101 WRITE (3,26) I, J, Y
024           WRITE (3,24)
025        24 FORMAT (35HOTHE PRODUCT OF A AND B MATRICES IS)
026           JOB=0
027           KET=0
      C ................. MULTIPLICATION OF A AND B MATRICES
028           DO 102 I=1,M
029           DO 102 L=1,K
030           V=0.
031           DO 102 J=1,N
032           IND=(I-1)*N+J
033           FIND (9'IND)
034           READ (9'IND) X
```

```
035         IND=M*N+(J-1)*K+L
036         FIND (9'IND)
037         READ (9'IND) Y
038         V=V+X*Y
039         KET=KET+1
040         IF (KET.NE.N) GO TO 102
041         KET=0
042         JOB=JOB+1
      C ..... STORING OF PRODUCT MATRIX ON THE DISK
043         IND=M*N+N*K+ JOB
044         FIND (9'IND)
045         WRITE (9'IND) V
046         WRITE (3,26) I, L, V
047    102  CONTINUE
048         END
```

```
            LOADER     RUN
```

```
$EXECUTION
```

```
THE MATRIX A IS
  1    1        3.00
  1    2        2.00
  1    3        5.00
  1    4        1.00
  2    1        4.00
  2    2        1.00
  2    3        6.00
  2    4        3.00
  3    1        4.00
  3    2        1.00
  3    3        2.00
  3    4        3.00

THE MATRIX B IS
  1    1        2.00
  1    2        1.00
  1    3        3.00
  1    4        4.00
  1    5        2.00
  2    1        3.00
  2    2        1.00
  2    3        2.00
  2    4        3.00
  2    5        2.00
  3    1        1.00
  3    2        5.00
  3    3        6.00
  3    4        2.00
  3    5        4.00
  4    1        2.00
  4    2        5.00
  4    3        1.00
  4    4        2.00
  4    5        3.00

HE PRODUCT OF A AND B MATRICES IS
  1    1       19.00
  1    2       35.00
  1    3       44.00
  1    4       30.00
  1    5       33.00
  2    1       23.00
  2    2       50.00
  2    3       53.00
  2    4       37.00
  2    5       43.00
  3    1       19.00
  3    2       30.00
  3    3       29.00
  3    4       29.00
  3    5       27.00
```

PROGRAM D

This program illustrates the use of the P-specification and slashes in a FORMAT statement. This program contains two data cards. Each data card contains four numbers, each of which has two digits to the left of the decimal point. The first specification in the FORMAT statement 11 moves the decimal point in the first number one place to the left and the second specification moves the decimal point in the second number one place to the right. The third and fourth numbers are read in as they appear on the data card. The sum of the first two numbers is labeled X and the sum of the last two numbers is labeled Y.

The list of the WRITE statement contains six items, A, B, X, C, D, and Y. Ordinarily these items are printed on a single line but, due to the slashes between the specifications in the FORMAT statement, each item is printed on a separate line. The blank lines between numbers on the output are caused by the double slash. The double slash at the end of the FORMAT statement causes the double blank lines between the first and the second sets of numbers.

Input:

22.215	31.217	12.321	12.325
14.321	41.214	13.214	54.102

```
                              FORTRAN   RUN
FORTRAN COMPILATION  VER 2 MOD 2
$NO MULTIPLY DIVIDE
$NO DICTIONARY
      C ...........P-CONVERSION AND SLASHES
      C
001           DO 21 I=1,2
002           READ (1,11) A, B, C, D
003      11   FORMAT (1PF10.2, -1PF10.2, OPF10.2, F10.2)
004           X=A+B
005           Y=C+D
006      21   WRITE (3,12) A,B,X,C,D,Y
007      12   FORMAT (F10.3/F10.3//F10.3//F10.3/F10.3//F10.3//)
008           STOP
009           END
```

```
                    LOADER     RUN
```

```
$EXECUTION
```

```
   2.222
 312.170

 314.392

  12.321
  12.325

  24.646


   1.432
 412.140

 413.572

  13.214
  54.102

  67.316
```