FORTRAN IV

REFERENCE SERIES

TYMSHARE™

TYMSHARE MANUALS

REFERENCE SERIES

FORTRAN IV

September 1968

Tymshare, Inc.
745 Distel Drive
Los Altos, California 94022

334 East Kelso Street
Inglewood, California
90301

464 Hudson Terrace
Englewood Cliffs, New Jersey
07632

Please send all comments about this manual to:
Library & Documentation Department, Tymshare, Inc.
925 East Meadow Drive, Palo Alto, California 94303

CONTENTS

CONTENTS (Continued)

CONTENTS (Continued)

CONTENTS (Continued)

CONTENTS (Continued)

CONTENTS (Continued)

CONTENTS (Continued)

SECTION 1

INTRODUCTION

Tymshare FORTRAN IV is composed of: 1) FORTRAN IV source language statements with which the actual programs are written, and which are upward compatible with most FORTRAN IV compilers; and 2) The Conversational Compiler System (CCS) command mode which controls the operating system under which FORTRAN IV language statements are used on the Tymshare system.

The FORTRAN IV Language statements are compiled and analyzed for errors one at a time as they are being entered. If the statement is incorrect, a diagnostic is given immediately. The statement then may be corrected. After modification, the program can be listed on the terminal, saved on a disk file, and/or executed. The user can specify a single statement or a range of statements to be executed as well as execute the entire program. Program execution can be interrupted at any time, and direct statements can be entered for immediate execution. Program execution can be resumed at the point of interruption.

Source programs in FORTRAN IV can be entered directly from the terminal, from a paper tape, or from a file. The user with a source program punched on cards can request that the cards be put on a disk file at the computer center, and then load the program from the file.

Data can be read in from the terminal, or from a file. Data on paper tape, magnetic tape, and cards may be read onto a disk file and then used. Similarly, results can be printed on the terminal or saved on a file. Large amounts of output can be printed on the high speed printer at the user's request at the computer center.

In addition to being conversational, Tymshare FORTRAN IV is compatible with other FORTRAN IV systems. Other advantages of Tymshare FORTRAN IV include:

- Editing and debugging features provided by CCS which make it easy to write and debug a program on-line.
- Large programs handled through program segmentation.
- Recursive calls to subprograms (functions, subroutines, segments)
- Dynamic storage allocation (subroutines, segments)
- Dynamic parameters in subroutines

FORTRAN IV should be used when any of the following characteristics are desired:

- When the user wishes to do complex arithmetic

- When the user wishes to have double precision
- When the user wants to use an existing FORTRAN IV program from other computers; or to debug a FORTRAN IV program to be run on other computers by taking advantage of the fact that FORTRAN IV is a common language.
- When the user wishes to use local names to isolate subroutines.
- When the user wishes to use long names for greater readability.

Purpose Of The Manual

This manual can be used both as a Tymshare tutorial guide and as a reference manual for FORTRAN IV. One of its aims is to provide sufficient instruction for a beginner in computer programming to write complete programs in FORTRAN IV and run them successfully on the Tymshare system.

This manual is organized so that an experienced programmer can skip Section 2 - An Introduction To FORTRAN IV Language Programming - and proceed directly to Section 3 which describes the complete FORTRAN IV language. Section 4 describes the commands in CCS; that is, how programs are entered, stored, and executed in the Tymshare system. It includes also the CCS commands for control of running programs and for debugging. Section 5 contains some user aids.

Section 6 provides a handy reference of Tymshare FORTRAN IV commands and statements.

Section 7 gives some sample programs written in FORTRAN IV and executed on the system.

Appendix A contains instructions for planning for efficient use of computer storage, using memory allocation estimation; and Appendix B contains the internal representation of ASCII codes.

Procedures For Entering And Leaving The System And A Sample FORTRAN Program Entered And Executed On-Line

The following is a summary of the log in and log out procedures for the Tymshare system. The complete description of the keyboard and terminal may be found in "The Tymshare Terminal" folder. A complete description of the EXECUTIVE user commands is found in the Tymshare EXECUTIVE Manual, Reference Series.

To gain access to the Tymshare time sharing system, you must first log in. As soon as the connection to the Tymshare computer is made, the system will type:

PLEASE LOG IN:

Type a Carriage Return. The system replies with:

ACCOUNT:A3↵

Type your account number (A3 in this case) followed by a Carriage Return. The system then types:

PASSWORD: ↵

Type your password followed by a Carriage Return. The letters in the password do not print. The system next types:

USER NAME:JONES ↵

The user name JONES is followed by a Carriage Return. The system next asks for a project code.

PROJ CODE:K-123-K↵

K-123-K is a project code. NOTE: A project code is optional. If no project code is wanted, simply type a Carriage Return in response to the system's request.

After you have entered the requested information correctly, the system will type:

READY 4/8 11:20

-

You are now in the EXECUTIVE system and can call FORTRAN by typing FORTRAN followed by a Carriage Return. FORTRAN will reply with a > when it is ready to accept a command.

LOGGING OUT

To exit from the Tymshare system, you first must be in the EXECUTIVE, characterized by a dash in the left hand margin. To return to the EXECUTIVE system from FORTRAN type:

QUIT ↵

The EXECUTIVE dash will appear in the left-hand margin. Now type:

-LOGOUT↵

followed by a Carriage Return. The system then will type:

TIME USED 0:37:12
PLEASE LOG IN:

You now may disconnect the line, or let another user log in.

The following is a sample FORTRAN IV program entered from the terminal and executed.

```
PLEASE LOG IN:
ACCOUNT:   A3
PASSWORD:
USER NAME: MM
PROJ CODE: KL-456
READY  10/23  12:54
-FORTRAN
>10 *THIS PROGRAM COMPUTES BALANCE ON A SERIES OF LOANS
>20 10 ACCEPT "LOAN NUMBER = ",NUM
>30 ACCEPT "PRINCIPAL= $",PRIN,"RATE= ",R,"PAYMENT= $",PAY
>40 DISPLAY "MONTH              BALANCE"
>50 MONTH=1
>70 20 XINTEREST=PRIN*R/12
>80 PRIN=PRIN+XINTEREST-PAY
>85. IF (PRIN.LE.0) GO TO 10
>90 WRITE (1,7) MONTH,PRIN
>95. 7 FORMAT (I5,10X,"$",F8.2)
>100 MONTH=MONTH+1
>110 GO TO 20
>120 END
>RUN
LOAN NUMBER = 13556
PRINCIPAL= $1500.00
RATE= .08
PAYMENT= $125.00
MONTH              BALANCE
    1           $ 1385.00
    2           $ 1269.23
    3           $ 1152.69
    4           $ 1035.37
    5           $  917.28
    6           $  798.39
    7           $  678.71
    8           $  558.24
    9           $  436.96
   10           $  314.87
   11           $  191.97
   12           $   68.25
LOAN NUMBER =
INTERRUPTED BEFORE:
20. 10 ACCEPT "LOAN NUMBER = ",NUM

20. >QUIT

-LOG

TIME USED 0:11:56
```

SECTION 2

INTRODUCTION TO CCS FORTRAN PROGRAMMING

A. INTRODUCTION TO FORTRAN IV LANGUAGE ELEMENTS

A program is made up of the statements necessary to tell the computer how to solve a problem - how to accept data, how to process it, and how to return the results to the user.

To analyze a problem to be solved, we must first examine the data and decide what calculations are necessary to produce the desired results using the data available.

Let us compute the interest on a loan. The basic computation is to multiply the principal by the interest rate. It may be written as:

```
PRINCIPAL * RATE
```

This is a FORTRAN expression and its value could be assigned to a variable written in an assignment statement.

```
XINTEREST = PRINCIPAL * RATE      (* is the FORTRAN IV multiplication symbol.)
```

Assume that the interest is compounded monthly. The borrower makes a monthly payment to cover the interest charge and to reduce the principal. Only two assignment statements are necessary to make the necessary computation:

```
XINTEREST = PRINCIPAL * RATE/12      (/ is the FORTRAN IV division symbol)
BALANCE = PRINCIPAL + XINTEREST - PAYMENT
```

The first assignment statement computes the monthly interest charge; the second statement compounds the interest and deducts the payment to determine the balance due.

Before the computations can be made, however, the computer must accept the data; that is, it must have the values for PRINCIPAL, RATE, and PAYMENT. An ACCEPT statement is required to obtain the data.

```
ACCEPT PRINCIPAL, RATE, PAYMENT
XINTEREST = PRINCIPAL * RATE/12
BALANCE = PRINCIPAL + XINTEREST - PAYMENT
```

The ACCEPT statement is an input statement for reading data into a program from the terminal.

After the data is read, the assignment statements are executed. The results of the processing are then ready to be returned.

A DISPLAY statement is added to tell the computer to print the results.

```
ACCEPT PRINCIPAL, RATE, PAYMENT
XINTEREST = PRINCIPAL * RATE/12
BALANCE = PRINCIPAL + XINTEREST - PAYMENT
DISPLAY PRINCIPAL, RATE, PAYMENT, BALANCE
```

The DISPLAY statement is an output statement which directs the computer to write on the terminal the current values of the following list of variables:
PRINCIPAL, XINTEREST, PAYMENT and BALANCE.

At this point the statements provide all the instructions necessary to read, compute, and write the information for a single loan record.

Suppose that many loans must be processed. The program must loop; control must be transferred from the last of the sequence of statements back to the first. A transfer of this type can be made only to a statement having a statement number, which is a number chosen by the programmer and prefixed to the statement to which the transfer is to be made. Any number may be used.

Now a GO TO statement is used to transfer control.

```
100 ACCEPT LOAN-NUMBER, PRINCIPAL, RATE, PAYMENT
    XINTEREST = PRINCIPAL * RATE/12
    BALANCE = PRINCIPAL + XINTEREST - PAYMENT
    DISPLAY LOAN-NUMBER, PRINCIPAL, RATE, PAYMENT, BALANCE
    GO TO 100
```

The GO TO statement is one of the control statements in FORTRAN IV.

We have added LOAN-NUMBER to the list of variables to be read and written. Now with this program many loans with varying rates of interest can be processed.

To amplify the problem above, suppose that instead of processing many loans, the balance on a single loan is to be processed for all of the months for which payment is made and the results printed in the form of a chart.

Again, the program must loop, this time for each month's payment. The number of the month is given an initial value of one and then increased by one each time that the loop is executed.

In this program an additional check must be made to determine when the entire principal has been paid, that is, when the balance is less than or equal to zero (BALANCE .LE. 0). This check is made using a logical IF statement. If the logical expression (BALANCE .LE. 0) is true, the statement following it (GO TO 90) is executed and control is transferred out of the loop to the statement labeled 90. If the logical expression is false, that is, the balance is not less than or equal to zero, the statement GO TO 90 is ignored and the next statement in sequence (DISPLAY MONTH, BALANCE) is executed.

The fourth statement in the program (BALANCE = PRINCIPAL) initializes the variable BALANCE to the value of PRINCIPAL that was accepted as input. Each time through the subsequent loop, the new value of BALANCE is used to calculate XINTEREST.

The text enclosed in the quote marks will be printed on the terminal when the program is executed.

```
    ACCEPT "PRINCIPAL = ", PRINCIPAL, "RATE = ", RATE, "PAYMENT = ", PAYMENT
    DISPLAY "MONTH                              BALANCE"
    MONTH = 1
    BALANCE = PRINCIPAL
20  XINTEREST = BALANCE * RATE/12
    BALANCE = BALANCE + XINTEREST - PAYMENT
    IF (BALANCE .LE. 0) GO TO 90
    DISPLAY MONTH, BALANCE
    MONTH = MONTH + 1
    GO TO 20
90  STOP
    END
```

The program will now produce a chart which shows the monthly balance on the loan. This program is essentially the one first presented on page 4.

B. KEY STEPS IN PROGRAMMING

As shown above, a program can be written in a computer programming language such as FORTRAN IV to solve a problem. In this section we will outline the key steps in writing a program.

To write a program, the key steps are:

1. Defining the problem
2. Selecting a method for solution
3. Analyzing the problem
4. Writing the instructions in the language
5. Debugging and checking the program
6. Documenting the program

A brief description of each step is as follows.

1. Defining The Problem

Before you can write a program to solve a problem, you must know what the problem is. To define a problem you should consider:

a. What information is given: (Input)
b. What answers are required: (Output)
c. In what manner is the data supplied?
d. How are answers to be given and what accuracy is desired?
e. Are there any options?

2. Selecting The Method

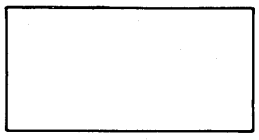
Usually there are several ways to solve a problem. Decide on one that is best for your particular problem. You may begin by preparing a brief step-by-step numerical solution to your problem using as many methods as you can. After some experience, it will be easier to see which method is better for computer solution.

3. Analyzing The Problem

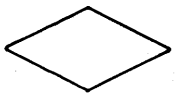
Once the method for solution is decided, you may decide on the steps for solving the problem. A program may be solved primarily by evaluating a sequence of formulas, or may require more involved steps. In the latter case, a detailed list of steps in mathematical form or by means of a flowchart is necessary. A flowchart is a pictorial representation of the problem and is used to illustrate the logical flow of steps necessary to solve the problem.

Conventionally:

Rectangular boxes are used for input, output and computations



Diamond-shaped boxes are used for decisions.



Circles are used as connectors when flowcharts become more complicated.

Figure 1 is an example of a simple flowchart for a program to find the sum of the reciprocals of n numbers.

Usually it is worthwhile to prepare test cases that exercise all options; that is, follow all paths of the flowcharts.. A test case should contain input data for which correct answers are known.

4. Writing The Instructions (Coding)

Writing the instructions usually is called coding, and is the actual writing of the program in a language "understood" by the computer. FORTRAN IV is a procedure-oriented rather than a machine-oriented language, and is made up of a small group of statement types. Each step in the problem solution frequently corresponds to a single FORTRAN IV statement. In forming these statements, the correct order and proper syntax of the language must be used. You may write down all instructions before typing them into the machine from the terminal. This allows you to do much of your initial thinking while not connected to the computer.

5. Debugging

Debugging is testing and checking a program. When you are entering the instructions from the keyboard, the interactive features of the Tymshare system can be a great help to you in finding and correcting errors in your program. Any faulty statement will be detected and printed with an error diagnostic. The first step in debugging is to make the program run on the machine. The next step is to make sure that the answers given are correct. After FORTRAN syntax errors are corrected and the program runs, it is advisable to test it using data for which the correct answer is known. The final checkout is of course continued satisfactory use of the program.

6. Documenting The Program

It is a good idea to include comments in the program to remind yourself of what the program does and what you have done in the different steps. After the program is debugged, document the program if you want to use it again.

Now that we have introduced you to programming, we will proceed to a simple course in FORTRAN IV, which will enable you to write and use programs of your own on the Tymshare system. We suggest that you try to run some simple programs on the terminal to see how it works. FORTRAN IV used on the Tymshare conversational system is very easy to use. You will find that the actual use of the computer is your most valuable learning experience, and the fastest way to get acquainted with the system.

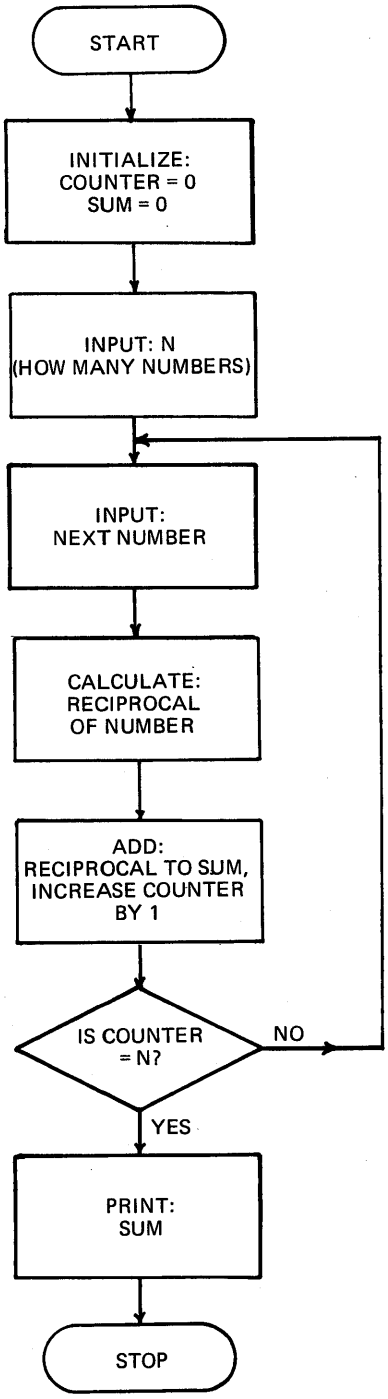


Figure 1 - Example Of A Flowchart

C. A SAMPLE FORTRAN PROGRAM EXECUTED IN CCS

We are going to take a sample problem and go through the programming steps to arrive at a solution. We will then show how the program is entered and executed on the terminal.

Problem

Write a general FORTRAN IV program to solve two simultaneous linear equations in two unknowns. In particular, give answers to X and Y in the following set of equations:

$$X + 2Y = -7$$
$$4X + 2Y = 5$$

The problem can be generalized as:

$$A_1 X + A_2 Y = B_1$$
$$A_3 X + A_4 Y = B_2$$

Different sets of two simultaneous linear equations could be solved by applying different values of B_1 and B_2 or different values of A_1, A_2, A_3, and A_4.

Solving for X and Y, the solution appears as follows:

$$X = \frac{(B_1A_4 - B_2A_2)}{(A_1A_4 - A_3A_2)} \qquad Y = \frac{(B_2A_1 - B_1A_3)}{(A_1A_4 - A_3A_2)}$$

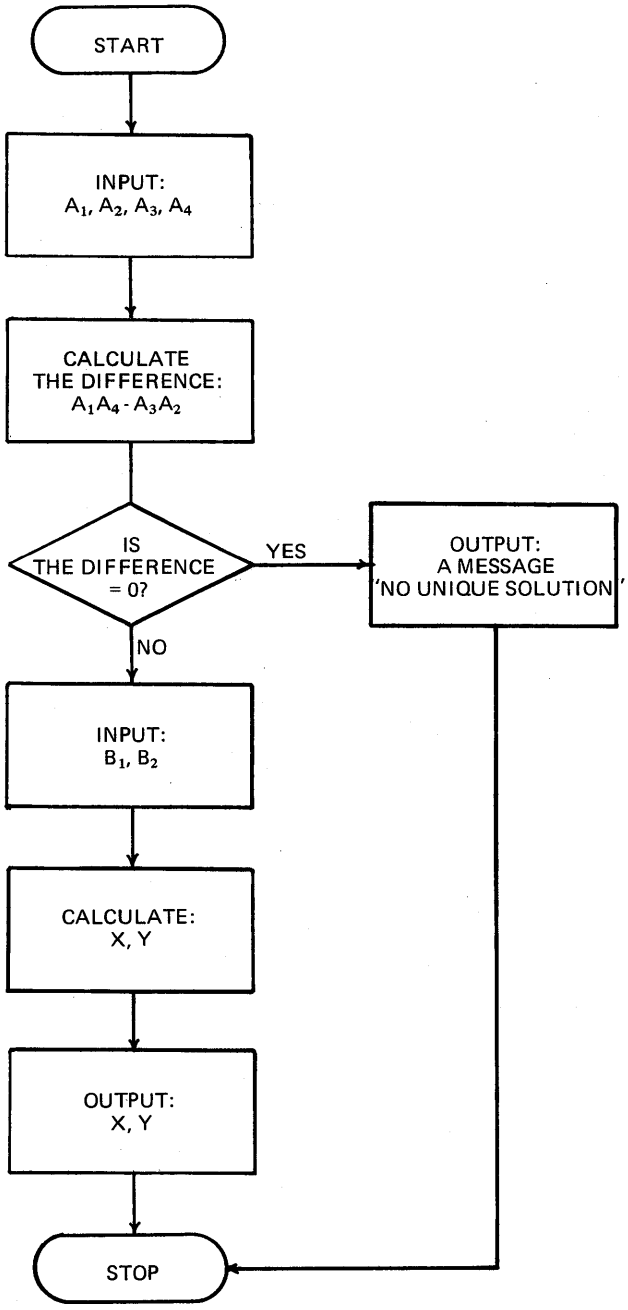
When the denominator $A_1A_4 - A_3A_2$ is equal to zero, there is no unique solution.

Coding

Here is the coding of the program:

```
10      ACCEPT A1, A2, A3, A4
15      DIFF = A1 * A4 - A3 * A2
20      IF (DIFF .EQ. 0) GO TO 100
25      ACCEPT B1, B2
30      X = (B1 * A4 - B2 * A2)/DIFF
35      Y = (B2 * A1 - B1 * A3)/DIFF
40      DISPLAY X, Y
45      GO TO 200
50  100 DISPLAY " NO UNIQUE SOLUTION"
55  200 STOP
60      END
```

Each statement corresponds to a step in the flowchart. Every statement has a line number (10, 15, 20, ...). The statements are executed in the order in which they are numbered unless a transfer occurs. Each line number identifies a line. Line numbers keep the program statements in order. As we shall see later, line numbers are used to refer to the program statements when modifying and insert-


START
INPUT:
A_1, A_2, A_3, A_4
CALCULATE
THE DIFFERENCE:
$A_1A_4 - A_3A_2$
IS
THE DIFFERENCE
= 0?
YES
OUTPUT:
A MESSAGE
'NO UNIQUE SOLUTION'
NO
INPUT:
B_1, B_2
CALCULATE:
X, Y
OUTPUT:
X, Y
STOP

ing statements. The choice of line numbers is arbitrary; they may be any number from .001 to 999.999, assigned either explicitly by the programmer or implicitly by the system.

There are two lines in the program with statement numbers in addition to line numbers (lines 50 and 55). These are statement numbers. Statement numbers are part of the FORTRAN IV language and are used as reference points by the program. Statement numbers can be any integer number and need not be in order.

Now we will go through the program step by step.

```
10      ACCEPT A1, A2, A3, A4
```

The first statement (10) is an ACCEPT statement for reading input data. The ACCEPT statement is unique to Tymshare FORTRAN IV and allows data to be read in a format-free form from the terminal. By format-free form we mean that the user simply types in the data values separated by a comma, a space, or a Carriage Return. The data is not input according to a specific format. When the program is executed, the system will ring a bell and wait for the user to type in the data values specified in the ACCEPT statement. A1, A2, A3, and A4 are variable names in which the values are to be stored.

```
15      DIFF = A1 * A4 - A3 * A2
```

The next statement (15) is an assignment statement. In this statement the value of the expression $A_1A_4 - A_3A_2$ is assigned to the variable DIFF. DIFF is another variable name. A variable name can be one or more alphabetic characters or digits, but the first character must be a letter of the alphabet. It is good practice in programming to choose a variable name that has some meaningful relationship to the program so it can be read easily. Notice two things: 1) multiplication is stated explicitly with a '*' sign. This is one of the arithmetic operators in FORTRAN IV. The others are '+' for addition, '-' for subtraction, '/' for division, and '↑' for exponentiation (raising a number to a power). 2) The correct form of an assignment statement is one in which the left side is the name of a single variable. The equal sign means "is to be replaced by" rather than "is equivalent to". This is an important distinction. For example, X = X + 1 is not an equation but a valid assignment statement.

```
20      IF (DIFF .EQ. 0) GO TO 100
```

In this statement we ask the question: "Is the difference equal to zero?" This type of statement is one of the control statements for making decisions. This statement allows you to branch to a different part of the program if the answer to the above question is "yes". This type of statement is an example of a logical IF statement. If the condition (DIFF equal to zero) is true, the statement following (GO TO 100) will be executed; otherwise GO TO 100 is not executed and the program goes on to the next statement. The statement GO TO 100 here is called the conditional GO TO statement. It

consists of the word GO TO and a statement number. It transfers operation to the part of the program specified by the statement number (in this case, statement 100), if and only if the condition specified is true.

.EQ. stands for "equal to" in FORTRAN IV, and must be surrounded by dots. .EQ. is one of the logical operators in the language. The others are not equal to (.NE.), less than (.LT.), less than or equal to (.LE.), greater than or equal to (.GE.), and greater than (.GT.). We call the expression DIFF .EQ. 0 a logical expression.

```
25      ACCEPT B1, B2
```

Line 25 is another ACCEPT statement to read in the values of B1 and B2.

```
30      X = (B1 * A4 - B2 * A2)/DIFF
35      Y = (B2 * A1 - B1 * A3)/DIFF
```

Statements numbered 30 and 35 complete the computation of the solution. Notice that the denominator is evaluated previously as the variable DIFF and is used in these two statements. Notice also that parentheses are used to specify that the numerator is the entire quantity (B1 * A4 - B2 * A2). FORTRAN distinguishes between the quantities $A - \frac{B}{C}$ and $\frac{A - B}{C}$. The first is written as A - B/C, the second as (A - B)/C. Parentheses may be used to specify the order of operations in expressions. If parentheses are omitted, the order of arithmetic operations is ↑, * and /, + and -.

```
40      DISPLAY X, Y
```

The next statement (40), a DISPLAY statement, is to output answers. When the statement is encountered, the values of variables X and Y are printed on the terminal. The DISPLAY statement is for format-free output just as the ACCEPT statement is for format-free input. This means that the system supplies the format that is used for output. Format-free input and output statements make it very easy to input and output data.

```
45      GO TO 200
```

Line 45 is another control statement. It is an unconditional GO TO statement which transfers the program to statement 200. We put in this transfer because, after displaying the solutions to X,Y, we simply want to end the program. This transfer allows us to bypass execution of the next statement and go on to the end of the program.

```
50 100 DISPLAY "NO UNIQUE SOLUTION"
```

The next statement has a statement number (in addition to a line number). Earlier in line 20 we have told the program to transfer to this statement if the difference (A1 * A4 - A3 * A2) is zero. Any number can be used as a statement number. This statement prints a message NO UNIQUE SOLUTION. Any message (text) inside a pair of double quote marks can be printed with a DISPLAY statement.

```
55 200 STOP
```

The STOP statement stops execution of the program. Notice that it also has a statement number. Earlier in the program we have asked for a transfer to here after displaying the results.

```
60      END
```

The last statement is the END statement. Every Tymshare FORTRAN IV program must end with an END statement. The END statement may not be labelled with a statement number; that is, we cannot transfer to this statement from other parts of the program.

Before we go on to show you how to run this program on the computer, we will summarize what we have learned about FORTRAN IV from this example.

1. ACCEPT statements are used to input data values used for computation.
2. DISPLAY statements are used to output the results of computation, as well as any literal text.
3. Variables are used to identify the values of input data and to store computational results.
4. The computer evaluates expressions which are made up of variables and operators. Parentheses may be needed to specify the order of computation.
5. An assignment statement stores the value of an expression in a single variable.
6. Control statements allow the user to override the normal sequential order of execution in the program. Here we have seen two control statements; namely, the logical IF with a conditional GO TO, and the unconditional GO TO statement. Logical operators and logical expressions are used with logical IF statements.
7. Statement numbers such as 100, 200 are used to label statements in the program.
8. A STOP statement stops execution of the program.
9. The last statement of a FORTRAN IV program must be an END statement.

To execute the program on the computer, log in and call FORTRAN. When FORTRAN is ready to receive instructions, it prints the > symbol. Then you may start typing in the statements of your program with line numbers, ending each line with a Carriage Return. If you type an incorrect statement, the computer will ring a bell and print out the statement in error with an ↑ pointing to the mistake. Just retype the statement correctly. When you have finished typing all the statements, the computer will type a > and the program is ready to be executed.

To start execution of a program, type RUN or EXECUTE followed by a Carriage Return.

The following is the program typed on line and executed.

```
PLEASE LOG IN: A3;MM;

READY  9/25  9:36
-FORTRAN

>10 ACCEPT A1,A2,A3,A4

>15 DIFF=A1*A4-A3*A2

>20 IF (DIFF .EQ.0) GO TO 100

>25 ACCEPT B1,B2

>30 X=(B1*A4-B2*A2)/DIFF

>35 Y=(B2*A1-B1*A3)/DIFF

>40 DISPLAY X,Y

>45 GO TO 200

>50 100 DISPLAY "NO UNIQUE SOLUTION"

>55 200 STOP

>60 END


>RUN
2,5,9,4
7,8
   0.3243243243     1.27027027

STOPPED AT: 55
```

When execution of the program is completed, the message STOPPED AT 55 is printed, 55 being the last statement executed before END was encountered.

To save a debugged program for future use, type SAVE/File Name/↵. The file name can be any name you want to give the program. After we have executed the above program, we could do the following:

```
>SAVE /EQUATION/↵
NEW FILE
```

The computer tells you whether it is a NEW FILE or an OLD FILE. Saving the program on an old file will erase the contents of that file and replace them with the current program.

Type a Carriage Return after NEW FILE. Type a Carriage Return after OLD FILE if you do not want to keep the file. If you want to keep the old file, type ALT MODE or ESCAPE and choose another file name. The entire program is now saved with line numbers on the file named /EQUATION/, and a > is given.

To use this program in the future type:

```
>LOAD /EQUATION/
```

The computer will say O.K. and start loading. When it has finished, it will type a >. Now you can execute your program again.

Suppose you want to have a listing of the program in the file /EQUATION/. After you have loaded the file, type LIST↵ as shown in the following example.

```
>LOAD /EQUATION/
OK.

>LIST

 10.          ACCEPT A1,A2,A3,A4
 15.          DIFF=A1*A4-A3*A2
 20.          IF (DIFF .EQ.0) GO TO 100
 25.          ACCEPT B1,B2
 30.          X=(B1*A4-B2*A2)/DIFF
 35.          Y=(B2*A1-B1*A3)/DIFF
 40.          DISPLAY X,Y
 45.          GO TO 200
 50.      100 DISPLAY "NO UNIQUE SOLUTION"
 55.      200 STOP
 60.          END

>
```

The program is printed neatly on the terminal with line numbers. The statement numbers are aligned and the statements indented.

The commands RUN, SAVE, LOAD, LIST are part of Tymshare FORTRAN IV commands. They are not parts of the FORTRAN language but are operating features of the Tymshare Conversational Compiler System (CCS). You will learn about the other CCS commands later in the manual.

D. A SECOND EXAMPLE TO SHOW THE USE OF OTHER FORTRAN FEATURES AND CONVERSATIONAL COMMANDS

In the following example we will introduce you to some other FORTRAN language elements as well as other Tymshare FORTRAN CCS commands.

Problem

Write a program to calculate the mean and standard deviation of N numbers, where N is greater than 1. Print also the normalized data.

Input: Required are:

N - the number of observations

Ai through An - the observations

Compute mean and standard deviation:

$$\text{Mean} = \frac{\sum_{i=1}^{n} Ai}{N}$$

$$\text{Standard Deviation} = \sqrt{\frac{\sum_{i=1}^{n} Ai^2 - \frac{\left(\sum_{i=1}^{n} Ai\right)^2}{N}}{N - 1}}$$

$$\text{Normalized data} = \frac{Ai - \text{Mean}}{\text{Standard Deviation}} \qquad i = 1,2,\ldots,n$$

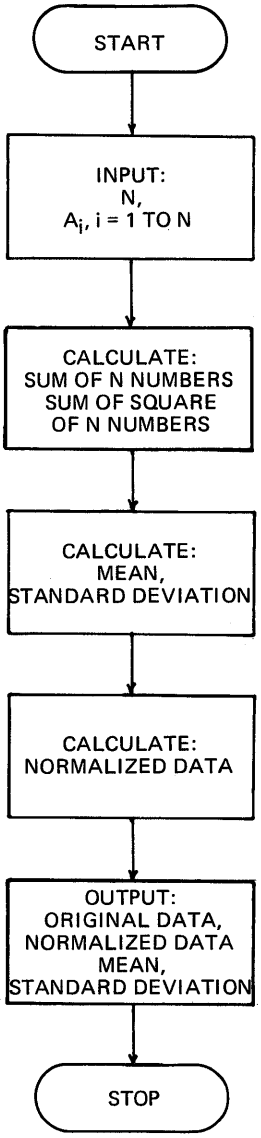
Output:

Original data, normalized data, mean, and standard deviation.

A(1) - A(N) NA(1) - NA(N) MEAN STD

Program Coding

```
 10    DIMENSION A(15)
 20    REAL N, MEAN, NA(15)
 30    ACCEPT "NUMBER OF OBSERVATIONS",N
 40    ACCEPT (A(I),I=1,N)
 50    SUM=0.
 60    SUMSQ=0.
 70    DO 20 I=1,N
 80    SUM=SUM + A(I)
 90    SUMSQ=SUMSQ + A(I)↑2
100 20 CONTINUE
```


START
INPUT:
N,
A_i, i = 1 TO N
CALCULATE:
SUM OF N NUMBERS
SUM OF SQUARE
OF N NUMBERS
CALCULATE:
MEAN,
STANDARD DEVIATION
CALCULATE:
NORMALIZED DATA
OUTPUT:
ORIGINAL DATA,
NORMALIZED DATA
MEAN,
STANDARD DEVIATION
STOP

```
110     MEAN = SUM/N
120     STD = SQRT[(SUMSQ-SUM↑2/N)/(N-1)]
130     DO 50 I=1,N
140     NA(I) = (A(I)-MEAN/STD
150 50  CONTINUE
160     DISPLAY "INPUT DATA",A
170     DISPLAY "NORMALIZED DATA",NA
180     DISPLAY "MEAN=", MEAN, "STAND. DEV. =", STD
190     END
```

In this program we are reading the N data values into a subscripted variable or an array, A. Thus A(1) is the first data observation and A(N) the Nth data observation. Storing data into an array makes it possible to refer to the list of observations by a single name.

To ask the computer to reserve space for the array A we use a DIMENSION statement as is done in line 10. Fifteen spaces are reserved for array elements A(1) to A(15) in the array A (for input data).

Line 20 is a type declaration statement, specifying that the variables N and MEAN are real numbers and the array NA contains real data (numbers with decimal points). This is necessary because FORTRAN distinguishes between integer numbers and real numbers. Thus 3/2=1, but 3./2.=1.5 in FORTRAN. Any variable name starting with I through N inclusive is treated as an integer variable unless specified to be a real number as we have done here. If this statement is left out, integer divisions will be performed and integer results will be given. The array NA is dimensioned using implied dimensioning in the type declaration statement.

Line 30 is an ACCEPT statement with descriptive text in quote marks. The text will be printed on the terminal when the program is executed. Line 40 reads in the N data values and stores them in the array A.

Lines 50 to 100 correspond to box 2 of the flowchart. These statements sum the N numbers and the square of the N numbers. Lines 50 and 60 initialize the variables SUM and SUMSQ to zero. The statement, DO 20 I=1,N controls the repetition of all succeeding statements down through and including the statement labeled 20. Repetition of these statements is controlled by varying an index called I, from an initial value of 1 to a terminal value of N in increments of 1. Lines 70 to 100 create a DO loop which sums the N numbers and the squares of the N numbers; each time adding the present value or square to the variables SUM and SUMSQ, the previously cumulated values. The CONTINUE statement on line 100 indicates that the process is to be continued.

After we calculated the sum $\left(\sum_{i=1}^{n} A_i\right)$ and the sum of the squares $\left(\sum_{i=1}^{n} A_i^2\right)$, we can proceed to calculate the mean and standard deviation. Line 110 calculates the mean. In calculating the standard deviation on line 120, we use the library function SQRT to find the square root. The expression (argument) is surrounded by square brackets. SQRT is one of the many built-in functions provided by the FORTRAN library.

Lines 130 through 150 correspond to box 4 in the flowchart. Another DO loop is used to normalize each of the N data values.

The last three statements of the program are output statements. Literals within quote marks are displayed describing what the output is in each case.

Before we run this program on the computer we will summarize the FORTRAN IV features in this example.

1. Arrays are dimensioned in DIMENSION statements.
2. Specific types of variables can be declared with a type declaration statement. Arrays may be dimensioned in a type declaration statement.
3. ACCEPT and DISPLAY statements can contain descriptive text as well as variables.
4. All variables must be defined when used on the right side of a replacement statement; for example, SUM (line 50) must be initialized (to zero in this case) before it can be used in line 80.
5. A DO loop is used for a repetitive process.
6. The FORTRAN library of functions can be used to simplify computation. The argument must be enclosed in square brackets.

Tymshare FORTRAN IV provides commands which make it easy to create programs on the terminal. These commands include line prompting, debugging, and editing features.

If you do not want to type in line numbers when you are first entering a program, ask the computer to prompt you with line numbers.

To do this, type in a line number followed by an increment in parentheses. (An implied ENTER command; the word ENTER is optional.) The computer will prompt you with line numbers starting with the number you specify increasing each time at the increment specified. A Carriage Return is necessary at the end of every statement. When the last statement of the program has been entered, you must type a Control D (D^C) to terminate the ENTER command.

Many editing features are available to you when entering a statement. For example, use a Control A (A^C) to delete the last character typed.

If you have entered a faulty statement, the computer will ring a bell when you hit the Carriage Return and print out the statement with an ↑ pointing to the first character that caused the error. This statement becomes the old line for editing purposes and any of the control characters described in this manual may be used.

In creating the following program, we will show you how to use some of these features.

```
>10(5)
10. DIMENSION A(5)
15. ACCEPT "NUMBER OF OBSERVATIONS  ",N
20. ACCEPT (A(I),I=1,N)
25. SUM=0.
30. SUMM←SQ=0.
35. DO 20 I=1,N
40. SUM=SUM+A(I)
45. SUMSQ=SUMSQ+A(I)↑2
50. 20 CONTINUE
55. MEAN=SUM/N
60. STD=SQRT[(SUMSQ-SUM*SUM/N/(N-1)[
```

This command caused the computer to prompt with line numbers beginning with line 10 in increments of 5.

The user typed another M instead of S. He typed a Control A(A^C) to erase the M, then typed the rest of the line.

```
60. STD=SQRT[(SUMSQ-SUM*SUM/N/(N-1)[
                                   ↑
60. STD=SQRT[(SUMSQ-SUM*SUM/N)/(N-1)]
65. DO 50 I=1,N
70. NA(I)=(A(I)-MEAN)/STD
75. 50 CONTINUE
80. DISPLAY "INPUT DATA",A
85. DISPLAY "NORMALIZED DATA",NA
90. DISPLAY "MEAN= ",MEAN, "STAND.DEV.=",STD
95. END
100.

>12 REAL N,MEAN
```

This statement is in error. The computer rang a bell and typed out the line with an ↑ pointing to the error. The user typed a Control Z(Z^C) followed by N. Line 60 up to the first N is copied. He typed in the rest of the line.

When the computer prompted line 100, the user typed a Control D(D^C) to end the line prompt. The user had forgotten to declare N, MEAN to be real variables, so he simply typed in the statement as line 12.
This statement will be inserted between lines 10 and 15.

So far the program has been entered. Now it is ready for execution. When an error is detected during execution, an error message is printed together with the statement that caused the error; execution will be terminated. At this point you may enter direct statements for execution to find out what caused the error. A direct statement is preceded by an @ sign. The statement can be entered when a > is given. A Direct statement, once executed, is discarded and is not part of the program. Once the error is detected and corrected, the program may be executed again.

In the following examples, the program just entered is to be executed.

```
>RUN
NUMBER OF OBSERVATIONS  10
1,2,3,4,5,
SUBSCRIPT OUT OF RANGE
20. ACCEPT (A(I),I=1,N)
```

The user tried to execute the program.

The computer typed out an error message and the statement in error after 5 data values were entered, then it typed 20. >.

```
20. >@DISPLAY I
   6
```

The user typed in @ for immediate execution of the statement: DISPLAY I

```
20. >DEFINITIONS A

10. DIMENSION A(5)
```

Then he typed the CCS command DEFINITIONS A which listed out the statement in which A is dimensioned. He found out that he had not reserved enough space for the array.

```
20. >10 DIMENSION A(10)
```

The computer continued to prompt 20. >. The user typed line 10 again to re-dimension A. The old line 10 is replaced by this line.

```
>RUN
NUMBER OF OBSERVATIONS  10
1,2,3,4,5,6,7,8,9,10

SUBSCRIPTED VARIABLE NOT DIMENSIONED
70. NA(I)=(A(I)-MEAN)/STD

>12 REAL N,MEAN,NA(10)
```

The user tried to execute the program again.

Another error is detected, the line that caused the error is listed. The user realized he had not dimensioned the array NA. He retyped line 12 including NA as a real array.

```
>RUN
NUMBER OF OBSERVATIONS  10
1,2,3,4,5,6,7,8,9,10
INPUT DATA   1.   2.   3.   4.   5.   6.   7.   8.   9.   10.
NORMALIZED DATA   -1.486301082   -1.156011953   -0.8257228238
   -0.4954336943   -0.1651445647   0.1651445647   0.4954336943
   0.8257228238   1.156011953   1.486301082
MEAN=    5.5STAND.DEV.=   3.027650354

STOPPED AT:  90.
```

He ran the program again, this time successfully.

Two things should be noticed. First, the command DEFINITIONS is used to type out the dimension statement for A. It is one of the commands provided by the CCS system to help you debug your program. Secondly, when the user types in line 10 again, it replaces the old line 10.

If you want the output to be in a neat form, you may use a formatted output.

A formatted output for mean and standard deviation may be:

```
WRITE (1,100) MEAN, STD
100 FORMAT ("MEAN=",F10.3,"STD=", F6.3)
```

The 1 in the WRITE statement is the file number and tells the computer that the values of the variables are to be written on the terminal (1 is reserved for terminal output). 100 is an arbitrary

number used to identify the format used. The values of the variables MEAN and STD are to be printed.

The FORMAT statement has a statement number corresponding to the one specified in the WRITE statement. Descriptive text to be printed is to be enclosed in quotes. F10.3 specifies the format in which MEAN is to be printed. The F-format prints a real number. F10.3 causes a maximum of ten characters (including the decimal point) to be printed, with three decimal places. STD is printed with the next format F6.3. STD is printed with a maximum of six characters with three digits after the decimal.

The formatted output for writing the data and the normalized data of the above may be

```
      WRITE (1,200) (A(I),NA(I),I=1,N)
  200 FORMAT (F10.3,5X,F10.3)
```

Data to be printed is A_i and NA_i for N values of each.

Each element of A and NA is to be printed as a real number with three places after the decimal point. 5X gives five spaces between the two numbers.

The program may be changed as follows to use formatted output. The user typed in the implied ENTER command 80:90, which allowed him to enter the statements. The computer prompts with an @ sign at the beginning of each line. The statements entered will be numbered uniformly between 80 and 90 when a terminating D^C is given. NOTE: The new lines will replace any old lines in that range.

```
>80:90
@DISPLAY "  INPUT DATA     NORMALIZED DATA"
@WRITE(1,200) (A(I),NA(I),I=1,N)
@200 FORMAT(F10.3,5X,F10.3)
@WRITE(1,300),MEAN,STD
@300 FORMAT("MEAN=", F6.2,2X, "STAND.DEV .=",F6.3)
@D^C
```

He typed a Control D (D^C) to tell the computer that he has finished entering statements. The computer prompted a >. He listed out the statements in this range.

```
>LIST 80:90

 80.          DISPLAY "  INPUT DATA     NORMALIZED DATA"
 82.          WRITE(1,200) (A(I),NA(I),I=1,N)
 84.      200 FORMAT(F10.3,5X,F10.3)
 86.          WRITE(1,300) MEAN,STD
 88.      300 FORMAT("MEAN=", F6.2,2X, "STAND.DEV .=",F6.3)
```

Now, when the program is re-run, the output is aligned in the form that the user wished it to be.

```
>RUN
NUMBER OF OBSERVATIONS  10
1,2,3,4,5,6,7,8,9,10
  INPUT DATA     NORMALIZED DATA
     1.000          -1.486
     2.000          -1.156
     3.000          -0.825
     4.000          -0.495
     5.000          -0.165
     6.000           0.165
     7.000           0.495
     8.000           0.825
     9.000           1.156
    10.000           1.486
MEAN=  5.50  STAND.DEV .= 3.027

STOPPED AT: 88.

>
```

Formatted input also may be used for reading in the data. Suppose in this example that the data comes from a score sheet as follows:

11	75
8	64
9	87
10	59

where the double line is the decimal place. You may write the following statements to read the input with a format:

```
    READ (0, 500) (A(I), I=1,N)
500 FORMAT (F5.2)
```

To input data from the terminal, you merely type

```
 1175  864  987 1059
```

allowing five columns for each number. You need not type in the decimal point. Each number would be read in with the format F5.2, that is, five digits for each number, two of which are decimal digits.

Now the program is modified to use formatted input as follows:

```
>20 READ(0,500) (A(I),I=1,N)


>22 500 FORMAT(4F5.2)
```

New line 20 replaces old line 20. Line 22 is inserted between lines 20 and 25.

The program is executed:

```
>RUN
NUMBER OF OBSERVATIONS  4
 1175  864  987 1059
  INPUT DATA       NORMALIZED DATA
    11.750           1.179
     8.640          -1.206
     9.870          -0.262
    10.590           0.289
MEAN= 10.21  STAND.DEV. = 1.303

STOPPED AT: 88.

>
```

And finally, here is the listing of the entire program:

```
>LIST

 10.           DIMENSION A(10)
 12.           REAL N,MEAN,NA(10)
 15.           ACCEPT "NUMBER OF OBSERVATIONS  ",N
 20.           READ(0,500) (A(I),I=1,N)
 22.       500 FORMAT(4F5.2)
 25.           SUM=0.
 30.           SUMSQ=0.
 35.           DO 20 I=1,N
 40.           SUM=SUM+A(I)
 45.           SUMSQ=SUMSQ+A(I)↑2
 50.        20 CONTINUE
 55.           MEAN=SUM/N
 60.           STD=SQRT[(SUMSQ-SUM*SUM/N)/(N-1)]
 65.           DO 50 I=1,N
 70.           NA(I)=(A(I)-MEAN)/STD
 75.        50 CONTINUE
 80.           DISPLAY "  INPUT DATA        NORMALIZED DATA"
 82.           WRITE(1,200) (A(I),NA(I), I=1,N)
 84.       200 FORMAT(F10.3,5X,F10.3)
 86.           WRITE(1,300) MEAN,STD
 88.       300 FORMAT("MEAN=",F6.2,2X,"STAND. DEV.=",F6.3)
 95.           END


>
```

SECTION 3

FORTRAN IV LANGUAGE

A. STATEMENT ELEMENTS

1. General

The FORTRAN IV language is composed primarily of combinations of constants, variable names, array names, together with the usual arithmetic, logical, and relational operators. There are certain rules for their formation and combination, which are presented here.

The result of the combination of constants and variables with arithmetic operators is an arithmetic expression. An arithmetic expression has a numeric value. The result of the combination of arithmetic expressions with relational operators, or of logical variables with logical operators, is a logical expression. A logical expression always has the value of .TRUE. or .FALSE.

2. Constants

A constant is a quantity which appears in a FORTRAN statement in numerical form.

Integer Constants

An integer constant is a positive or negative whole number up to eleven digits such as:

0

-1245

3859437

Real Constants

There are two forms of real constants, either with or without exponents.

Without exponents, the constant contains one or more digits and a decimal point (.) which must be written and may appear anywhere in the number. For example,

0.

3.14159265359

-0.07

.0000567

The magnitude of the constant ranges from 10^{-75} to 10^{75} or zero. The accuracy of a real constant is eleven significant digits.

With an exponent, the constant contains one or more digits with or without a decimal point, followed by the letter E, followed by the exponent. The exponent represents the power of ten by which the number to the left of the exponent is to be multiplied. This is called floating point or exponential representation.

Examples

5E-2	means	.05
1.E7	means	10^7
1.973E3	means	1973.
.0271828E+2	means	2.71828
25E3	means	25000.

The magnitude of the exponent is from E-75 to E+75.

The value of zero is represented simply as 0.

Complex Constants

A complex constant is expressed as two single precision numbers (eleven digits of accuracy) separated by a comma and enclosed in parentheses. The first number represents the real part of the complex number; the second represents the imaginary part of the complex number. A complex number always is printed with a decimal point.

Examples

(3, 5.2)	has the value of 3+5.2i
(-1.8, .16E2)	has the value of -1.8+16i
(2.4, 0)	has the value of 2.4+0i

Logical Constants

A logical constant can be either .TRUE. or .FALSE. .

Hollerith Constants

A Hollerith constant is expressed as nH followed by n characters, or as a group of characters enclosed by a pair of double quotes, a pair of single quotes, or a pair of $ signs. (The characters can be any character or digit except a semicolon and an up arrow.)

Examples

```
3HYES
5HABC12
7HAB  CDE
"XXX"
'CODE'
$YES$
```

3. Variables

Variable Types

A variable is a symbol or name which represents a quantity. The value of a variable may be changed throughout the program.

In CCS FORTRAN IV the name of a variable may consist of any number of alphanumeric characters. The first character must be alphabetic (A through Z).

There are five types of variables in CCS FORTRAN IV: Integer, Real, Double Precision, Complex, and Logical. The variable type corresponds to the type of data the variable represents. Thus, an integer variable represents integer data, a real variable represents real data, and so on. A number stored in a Double Precision variable has at least seventeen digits of precision.

The programmer must show the data type for each variable in his program. Data types other than integer and real must be declared explicitly with a "type declaration" statement. (See Section , Declaration Statements.)

Integer and real data types may be declared implicitly. If the first letter of a variable name is I, J, K, L, M, or N, the variable is Integer. Any other variable name not appearing in any type declaration statement is Real. This is implicit type declaration. Explicit type declaration in a type declaration statement however, overrides implicit type declaration.

Scalar Variables

A scalar variable represents a single quantity such as:

N
INDEX
ALPHA
X12

Array and Array Variables

A group of variables which form or belong to a single class or collection may be related to one another by subscript notation. Such a collection is called an array, and the variables belonging to the array are called array elements.

A string of numbers in a single row or column is thought of as a one-dimensional array. The elements are identified by a single subscript as shown below.

Usual Notation	FORTRAN Notation
a1	A(1)
a2	A(2)
.	.
.	.

.	.
ai	A(I)
.	.
.	.
.	.
an	A(N)

If two subscripts are used to identify the elements of an array, the array is a two-dimensional array. For example, if there are three rows and four columns in a table, A(2,3) refers to the element in the second row and third column.

In CCS FORTRAN IV, there is no limit to the number of dimensions of an array.

The array type may be Integer, Real, Double Precision, Complex, or Logical.

A subscripted variable represents a simple element of an array. A subscripted variable is denoted by a variable name followed by an arithmetic expression in parentheses (see "Expressions" below).

Examples

```
A(3)
B(-5)
C(0)
MOVE(X,5)
Y(M,1,1,N+3)
VOLTAGE (2*N+1,L,L+1)
```

4. Expressions

An expression in its simplest form may consist of a single constant, a variable, or a function (see "Functions"). Also, an expression may denote a computation between two or more constants and/or variables or functions.

There are two kinds of expressions in FORTRAN IV; arithmetic and logical. The value of an arithmetic expression is always an integer, real, or complex number. The value of a logical expression is either .TRUE. or .FALSE. .

a. Arithmetic Expressions

An arithmetic expression may consist of a single basic element; that is, a constant, variable, or function.

Examples

```
3.14        A(5)
X           SQRT[ALPHA]
```

More complicated arithmetic expressions may be formed by using arithmetic operators which determine the computations to be performed.

Arithmetic Operators

+	Addition
-	Subtraction
*	Multiplication
/	Division
** or ↑	Exponentiation

Unary arithmetic operations; that is, operations involving only one argument or operand, also are possible. For example,

-B means negative of B

+A means A

-C**2 means negative of C^2

Order of Operation

1. Parentheses may be used to specify the order of operation in an expression. When sets of parentheses appear within other sets of parentheses, the innermost set is evaluated first, then the next set, and so on.

2. When parentheses are not used, expressions are evaluated in the following order:

Exponentiation (** or ↑)

Unary minus (-)

Multiplication and Division (* and /)

Addition and Subtraction (+ and -)

3. Arithmetic expressions are evaluated from left to right. Thus, when two operators of the same precedence appear, the left operation is evaluated first.

Examples

A+B*C means $A+(B*C)$

A/B/C means $\frac{\frac{A}{B}}{C}$

A/B*C/D means $\frac{\frac{A}{B}*C}{D}$

A+B/C**2 means $A+\frac{B}{C^2}$

((A+B)/C)**2 means $\left(\frac{A+B}{C}\right)^2$

Modes of Expressions:Mixed Expressions

The kind of arithmetic performed depends on the type or modes of the operands. For example, if both operands are integers, integer arithmetic is used. Thus, 3/4 causes an integer division and has the value zero. But 3./4. gives a result of .75 since both operands are real and division with real numbers is performed.

Arithmetic expressions containing constants or variables of more than one type are called mixed expressions. In a mixed expression, as each operation is performed, the types of the two operands are compared; the smaller type is converted to the larger type.

Number Type	Magnitude
Complex	Largest
Double Precision	↑
Real	↕
Integer	↓
Logical	Smallest

Examples

A/I — The integer I is converted to real and real division is performed.

N/(I+2.) — The expression (I+2.) is first considered. Since 2. is real, I is converted to real and the result of the addition is real. N is then converted to real since it is to be divided by a real number. The result of the whole expression is real. If N=3 and I=4, the result is 0.5.

A*(N/2) — The result of the expression (N/2) is an integer since both operands are integers. (N/2) is then converted to a real number since it is to be multiplied by A which is real. If A=6 and N=3, the result is 6. .

A*(N/2.) — N is first converted to real since it is to be divided by a real number 2. . Real division is performed. Two real numbers are then multiplied together. If the values for A and N are the same as in the previous example, the result is 9. (3./2.=1.5 but 3/2=1).

B+1 — If B is a complex variable with a value of (2.7,8.1) and I=1, I will be converted to a real number (1.,0). The result will be (3.7,8.1).

D+B — If D is a double precision variable and B a complex variable, the value of D will be truncated to single precision, then treated as a complex number with zero as the imaginary part.

An exception to the conversion rule is with exponentiation. Integer expressions which appear as exponents are evaluated in integer mode regardless of what the other operand is. For example, a real operand may be raised to an integer power and the result is real.

b. Logical Expressions

A logical expression may consist of a single logical constant or a logical variable. The value of a logical expression is always a truth value, .TRUE. or .FALSE. .

More complicated logical expressions may be formed by using logical and relational operators. These expressions may be one of the following forms:

1. Relational operators combined with arithmetic expressions.
2. Logical operators combined with logical constants or logical variables.
3. Logical operators combined with either or both forms of the logical expressions described in 1. and 2. above.

Relational Operators

A relational operator makes a comparison between expressions. For example, the expression X .GT. Y means X is greater than Y. This expression has a value which at any time is either true or false. Comparisons may be made by means of any of the following relational operators.

Symbol	Customary Notation	Meaning
.EQ.	=	Equal to
.NE.	≠	Not equal to
.LT.	<	Less than
.LE.	≤	Less than or equal to
.GT.	>	Greater than
.GE.	≥	Greater than or equal to

Examples

```
A .EQ. B
X-5 .LT. Y+4
SQRT[BETA] .GE. ALPHA↑2
```

Unlike most FORTRAN IV compilers, CCS FORTRAN IV can make comparisons between arithmetic expressions of different types. For example, A .LE. I is valid. I will be converted to a real number before the relation is evaluated.

The relative magnitude of expression types and rules for comparing expressions are as follows:

Expression Type	Magnitude
Complex	Largest
Double	
Real	
Integer	
Logical	Smallest

Let E1 and E2 be arithmetic expressions of any type in the following:

(1) Comparing with .EQ. and .NE. :

E1 .EQ. E2

E1 .NE. E2

The expression type of smaller magnitude is converted to the type of larger magnitude. Comparison is then made. In the case of complex expressions, both real and imaginary parts are compared. For example, (1.,2.) .EQ. 1 is false. The integer 1 is converted to a complex number (1.,0) and the imaginary part of this number is not equal to 2. .

(2) Comparing with .LT., .GT., .LE., .GE. :

E1 .LT. E2

E1 .GT. E2

E1 .LE. E2

E1 .GE. E2

The expression type of smaller magnitude is converted to the type of larger magnitude as above. Take the difference between the two expression types of the same magnitude. In the case of complex expressions, compare the real part of the difference between E1 and E2 to zero. If the difference is positive, E1 is greater than E2. If the difference is equal to zero, E1 and E2 are equal. If the difference is negative, E1 is less than E2. For example, (1.,2.) .LT. (1.,3.) is false since the difference between the real parts is zero. (2.,-100) .GT. (1.,4.) is true since the difference between the real parts is positive.

Logical Operators

A logical operator operates on logical expressions. The result of a logical operation is .TRUE. or .FALSE. .

There are four logical operators. In the following table where X and Y are logical expressions, the results of the logical operations are shown.

OPERATOR	X Y	.TRUE. .TRUE.	.TRUE. .FALSE.	.FALSE. .TRUE.	.FALSE. .FALSE.
AND	X .AND. Y	.TRUE.	.FALSE.	.FALSE.	.TRUE.
Inclusive OR	X .OR. Y	.TRUE.	.TRUE.	.TRUE.	.FALSE.
Exclusive OR	X .EOR. Y	.FALSE.	.TRUE.	.TRUE.	.FALSE.
NOT	.NOT. X	If A is .TRUE., then .NOT. A is .FALSE.; if A is .FALSE., then .NOT. A is .TRUE. .			

Examples Of Logical Expressions With Logical Operators

A .AND. B. where A,B are logical variables.

(X .EQ. 3) .OR. Y .LE. 4

(E*5 .GT. A-B) .OR. (E .LE. 100)

(A/2 .LT. SQRT[5]) .AND. I .EQ. J

.NOT. W .AND. .NOT. L

(A .GT. 100) .EOR. (B .GT. 200)

Order of Computation In Logical Expressions

Where parentheses are omitted, the order in which operations are performed is as follows:

Evaluation of Functions (for example, SQRT)

Exponentiation (** or ↑)

Unary minus (negative)

Multiplication and Division (* and /)

Addition and Subtraction (+ and -)

Relational operators (.EQ., .NE., .LT., .LE., .GT., .GE.)

.NOT.

.AND.

.OR.

.EOR.

For example, the expression SQRT[A] .GT. C↑3 .AND. .NOT. R .OR. S is evaluated in the following order:

1. SQRT[A] call the result V (function)
2. C↑3 call the result W (exponentiation)
3. V .GT. W call the result X (relational operators)
4. .NOT. R call the result Y (highest logical operator)
5. X .AND. Y call the result Z (second highest logical operator)
6. Z .OR. S final operation

Parentheses may be used to modify the order of evaluation.

B. ARITHMETIC AND LOGICAL REPLACEMENT STATEMENTS

1. General

A replacement statement uses the replacement operator "=". A replacement statement has the form V=e meaning to replace the current value of the variable V with the current value of the expression e. The "=" means "is replaced by" rather than "is equivalent to".

2. Arithmetic Replacement Statements

The general form of an arithmetic replacement statement is variable = expression where V is a scalar or a subscripted variable and e is an arithmetic expression. Execution of the statement causes the value of the variable to be replaced by the value of the expression. If the mode of the expression differs from the mode of the variable, the mode of the expression is converted to the mode of the variable before replacement takes place.

Examples

In the following, V, A, and B are real variables; I and J are integer variables; C is a complex variable; and D is a double precision variable.

VAR=A*B/5	The current value of VAR is replaced by the result of A*B/5.
A=I	The value of I is converted to a real value and this result replaces the current value of A.
I=I+1	The value of I is replaced by the value I+1.
A=C	The real part of the complex variable C replaces the value of A.
C=A	The value of A replaces the value of the real part of the complex variable C, the imaginary part of C is set to zero.
C=I↑J	I is raised to the power J and the result is converted to a real value which replaces the real part of the complex variable E. The imaginary part of C is set to zero.
D=J	The value of J is converted to double precision and stored in D.
C=(3.4,2.0)	The value of C is replaced by the complex constant (3.4,2.0). Note that (C=A,B) where A and B are variables is not allowed.

3. Logical Replacement Statements

The general form of a logical replacement statement is variable = expression where the variable is a scalar or subscripted logical variable, and the expression is a logical expression. Execution of the statement causes the variable to be replaced by a value of .TRUE. or .FALSE. depending on whether the expression is true or false.

Examples

In the following, H, K, and M are logical variables and I is an integer variable.

H=.TRUE. The value of H is replaced by the logical constant .TRUE. .

K=.NOT. M If M is .TRUE., the value of K is replaced by the logical constant .FALSE. . If M is .FALSE., the value of K is replaced by the logical constant .TRUE. .

H=3 .EQ. I The value of I is converted to a real value. If the real constant 3. is equal to this result, the logical constant .TRUE. replaces the value of H. If 3. is not equal to this result, the logical constant .FALSE. replaces the value of H.

NOTE: If a logical expression is used in an arithmetic assignment statement, the logical expression has an integer value of 1 if it is .TRUE., and an integer value of zero if it is .FALSE. .

C. CONTROL STATEMENTS

1. General

Normally, FORTRAN statements are executed sequentially. That is, after one statement has been executed, the statement immediately following it is executed. Control statements allow the programmer to alter and control the normal sequence of execution of statements in a program. When it is necessary to alter this normal sequence, certain statements are labeled so that they may be referred to by control statements. A statement label can be any integer number. Any statement except declaration statements can be labeled.

2. IF Statements

Two types of IF statements are provided to make decisions; namely, logical IF and arithmetic IF. With a logical IF, decisions are based on a logical quantity being true or false. With an arithmetic IF, decisions are based on an arithmetic quantity being less than zero, zero, or greater than zero.

Logical IF Statements

The logical IF statement has the general form IF (expression) Statement where the expression is any logical expression and the Statement is any executable statement except DO or another logical IF statement.

If the expression is true, the statement is executed; execution then goes to the next statement except if the statement is a transfer statement. If the expression is false, the statement will not be executed; execution simply goes to the next statement. For example, in the following statements

```
IF(X .GT. Y) A=B
C=A*5
```

if the value of X is greater than Y, the value of A will be replaced by the value of B; then C is set to A*5. If X is not greater than Y, A will not be set to the value of B. C will be replaced by A*5 whatever the value of A is.

Arithmetic IF Statements

The arithmetic IF statement has the general form IF (expression) n1, n2, n3 where the expression is any arithmetic expression (except complex) and n1, n2, and n3 are statement numbers. This statement causes control to be transferred to the statement numbered n1, if the value of the expression is negative, to n2 if it is zero, and to n3 if it is positive.

Examples

IF (K-N) 10, 10, 20	If the expression K-N has a value which is less than or equal to zero, indicating that K is less than or equal to N, the next statement executed is statement 10; otherwise if K is greater than N, the next statement executed is statement 20.

```
IF (Y-A(I)) 10, 15, 15
```

If Y is less than A(I), that is, the value of Y-A(I) is less than zero, the statement executed next is statement 10. If the value for Y-A(I) is zero or positive, the statement executed next is statement 15.

```
IF (X(I,J)*3↑2) 12, 5, 30
   .
   .
   .
 5 M=I+J
   .
   .
   .
30 C=D-B
   .
   .
   .
12 E=(F*B)/C
   .
   .
   .
```

If the value of the expression (X(I,J)*3↑2) is negative, statement 12 is executed next. If the value of the expression is zero, statement 5 is executed next. If the value of the expression is positive, statement 30 is executed next.

3. DO Statements

In a majority of numerical computation and information processing procedures, repetitive processes are used. Computer programs almost always involve a group of steps that need to be executed repeatedly. The iteration or DO statement controls repetitive processes.

The statement `DO 30 J=1, 10, 1` controls the repetition of all succeeding statements down through and including the statement labeled 30. Repetition of these statements is controlled by varying an index called J from an initial value of 1 to a terminal value of 10 in increments of one.

The general form of a DO statement is `DO n I=m1, m2, m3`

where n is the statement number of the last step in the repetition.

I is the index variable whose value changes during the repetitive process. The name may be any non-subscripted variable.

m1 is the initial value for the index variable.

m2 is the upper limit for the index variable; the index variable must exceed this value to terminate the repetition.

m3 is the increment in the index variable for successive repetitions. (m3 may be omitted, in which case the increment is assumed to be one.)

NOTE: The indexing parameters m1, m2, and m3 may be any arithmetic expression. The increment m3 can be negative.

The group of statements starting with the DO statement, through and including the statement labeled n, is called a DO loop.

Example 1

In this example, the 'increment' is negative. The value of I is printed beginning with 3, by steps of -1, ending with 0.

```
>10 DO 30 I=3,0,-1

>20 30 DISPLAY I

>30 END

>RUN
   3
   2
   1
   0

STOPPED AT: 20.

>
```

Example 2

In this example, statement 100 is executed beginning with X=1, by steps of .2 and stops when X exceeds 1.5.

```
>10 DO 100 X=1,1.5,.2

>20 100 DISPLAY X

>30 END

>RUN
   1.
   1.2
   1.4
STOPPED AT:  20.
>
```

Throughout the DO loop, the index variable is available both in subscripts and as an ordinary integer variable. For example, the statements

```
    DO 100, I=3,20
100 A(I)=I
```

will set A(3) to a value of 3, A(4) to a value of 4, up to A(20)=20.

NOTE: The indexing parameters of a DO statement (I, m1, m2, m3) may not be changed by a statement inside a DO loop.

Example

The following DO loop is used to find the total of N numbers of the array X.

```
      SUM=0
      DO 25 I=1,N
   25 SUM=SUM+X(I)
```

A CONTINUE statement may be used as the last statement for a DO loop. The above may be written as

```
      DO 25 I=1,N
      SUM=SUM+X(I)
   25 CONTINUE
```

The CONTINUE statement causes no action; it merely serves as a dummy statement to refer to the end of the loop.

Nested DO Loops

A DO loop may include other DO loops provided that the range of each inner loop is contained completely within the range of the outer loop. That is, the two DO loops may not partially intersect one another. These loops are called nested loops. The following skeleton examples illustrate this:

Allowed

```
DO 10 I=1,N
DO 20 J=1,M
---
---
20 CONTINUE
10 CONTINUE
```

Not Allowed

```
DO 10 I=1,N
DO 20 J=1,M
---
---
10 CONTINUE
---
20 CONTINUE
```

Allowed

```
DO 15 K=1,N
---
---
DO  5 A=1,7.5,.5
---
---
5 X=7+Y
---
---
DO  7 BAJ=2,3
---
---
DO 100 B=A*C,2
---
---
100 CONTINUE
---
---
7 X=22
---
---
---
15 CONTINUE
```

Nested loops may end with the same terminal statement. The following statements sum up an array of three rows and four columns.

```
      TOTAL=0.
      DO 15 I=1,3
      DO 15 J=1,4
      TOTAL=TOTAL+A(I,J)
   15 CONTINUE
```

While it is permissible to transfer control out of a DO loop, transfer of control into a loop is not allowed. The following examples illustrate this:

Allowed

```
   DO 200 K=5,M,2
   C=K*P-Q
   IF(C .GT. 100) GO TO 50
200 CONTINUE
   ---
   ---
 50
```

Not Allowed

```
   DO 10 I=1,N
   ---
 5 ---
10 CONTINUE
   GO TO 5
```

One exception to this rule is when returning to a DO loop from an extended range. An extended range of a DO loop is statements outside the loop to which a statement within the DO loop transfers. They could be included as part of the loop but are written outside since they were to be executed again in the program. For example, in the following statements

```
   DO 100 K=1,10
   X=A*K
   Y=B*K↑2
   GO TO 50
 60 C=X-Y
100 CONTINUE
 50 X=SQRT[X*2]
   Y=SQRT[Y*3]
   GO TO 60
```

the statements

```
 50 X=SQRT[X*2]
   Y=SQRT[Y*3]
```

are in the extended range of the DO loop. Thus, GO TO 60 is permitted since this returns control to the DO loop from an extended range of the loop.

Implied DO Loops

An implied DO loop may be used in an input, output, or a DATA statement.

For example, the statement

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

reads in the data values for A(1) to A(10).

The statement

```
DISPLAY (B(I),I=1,10,2)
```

prints the data values for A(1), A(3),..., up to A(9).

The statement

```
DATA (A(I),I=1,3)/1,2,3/
```

initializes the A(1) to 1, A(2) to 2, and A(3) to 3.

4. GO TO Statements

The GO TO statements permit the user to alter the sequence in which program statements are executed. The GO TO statements transfer control to the statement number specified in the GO TO statement. There are three types of GO TO statements:

1. Unconditional GO TO statement.
2. The Computed GO TO statement.
3. The Assigned GO TO statement.

Unconditional GO TO Statements

The unconditional GO TO statement causes an unconditional transfer. The form of the statement is GO TO statement number

Example

```
GO TO 20
```

The GO TO statement alone is unconditional. However, the GO TO statement may be written so that execution of it is conditional. For example the statement IF (X .EQ. Y) GO TO 30 will cause transfer to the statement labeled 30 if the values of X and Y are equal. If X is not equal to Y, the next statement in sequence will be executed.

The following statements illustrate the use of GO TO statements. k1, k2 --- kn is a list of integers. The following statements count the number of positive even and the number of negative even integers. Zeros are ignored.

```
      NPOSEVEN=0
      NNEGEVEN=0
      DO 10 I=1,N
      IF (K(I) .NE. 0 .AND. (K(I)-K(I)/2*2) .EQ. 0) GO TO 20
      GO TO 10
   20 IF (K(I) .GT. 0) GO TO 30
      NNEGEVEN=NNEGEVEN+1
      GO TO 10
   30 NPOSEVEN=NPOSEVEN+1
   10 CONTINUE
```

Computed GO TO Statements

The computed GO TO allows transfer to one of the several places in the program, depending on the integer value of a variable. The form of the statement is GO TO (n1,n2,n3,...)I where I is a non-subscripted integer variable and n1, n2,... are statement numbers to which the program will transfer depending on the value of I. If the value of I is 1, the program will transfer to the statement labeled n1; if the value of I is 2, the program will transfer to statement labeled n2, and so on. For example, GO TO (60,70,85)K will transfer control to statement 60, if the value of the expression K is 1, to 70 if the value is 2, and to 85 if the value is 3.

If the value of K is not an integer, it will be truncated to an integer value. If K is negative or greater than the number of statement numbers, an error will be indicated.

Assigned GO TO Statements

The assigned GO TO statement is useful for conditional branching. The form of the statement is GO TO variable
This statement transfers control to the statement whose number was last assigned to the variable by an ASSIGN statement. The variable must appear in some previously executed ASSIGN statement.

Examples

```
GO TO L
GO TO EX (3)
```

The variable in an assigned GO TO statement is a control variable and has a statement number (a label) as its "value" and not a numerical quantity.

5. ASSIGN Statement

This statement assigns a statement number to a variable for a subsequent assigned GO TO statement. The form of the statement is ASSIGN integer TO variable where integer is the statement number to which control will be transferred by the assigned GO TO statement.

In the following

```
      .
      .
      .
      ASSIGN 10 TO R
      IF ((A .LT. 100) .AND. (B .LT. 50)) ASSIGN 20 TO R
      A=450
      B=-200
      GO TO R
   10 X=(A+B)*2
      Y=(A-B)*X
```

```
      GO TO 50
   20 X=3-B*A
      Y=3+B*A
      GO TO 50
      .
      .
   50 .
```

statement label 10 is assigned initially to R. If the value of A is less than 100 and B less than 50, R is reassigned to statement 20. When the statement GO TO R is executed, control will be transferred to statement 10 or statement 20 depending on the value of R.

Another Example

```
      .
      .
      .
      ASSIGN 100 TO A(1)
      ASSIGN 200 TO A(5)
      ASSIGN 450 TO A(9)
      I=M*N-J
      GO TO A(I)
      .
      .
      .
  100 .
      .
      .
  200 .
      .
      .
  450 .
      .
      .
```

When the statement GO TO A(I) is executed, the program will go to statement 100, 200, or 450 depending on whether the value of I is 1, 5, or 9 respectively.

6. CONTINUE Statement

The form is

```
CONTINUE
```

The CONTINUE statement is used as a dummy statement which may be placed anywhere in the program without affecting the sequence of execution.

It may be used as the last statement of a DO loop. A CONTINUE statement must be used to avoid ending the DO loop with a GO TO, PAUSE, STOP, RETURN, Arithmetic IF, or another DO statement.

Example 1

In the following statements which count the number of non-zero elements of the array A, the CONTINUE statement is used to skip the count if an element is zero.

```
      NCOUNT=0
      DO 15 I=1,N
      IF (A(I)-1)10,15,10
   10 NCOUNT=NCOUNT+1
   15 CONTINUE
```

Example 2

```
      DO 40 I=1,20
    8 IF (X(I)-Y(I))5,40,40
    5 X(I)=X(I)+1
      Y(I)=Y(I)-2
      .
      .
      .
      GO TO 8
   40 CONTINUE
```

In the above example, the CONTINUE statement is used as the last statement of the DO loop to avoid ending the loop with the statement GO TO 8.

7. PAUSE Statement

The form is

```
PAUSE
```

A PAUSE statement may be placed anywhere in a program. When this statement is executed, the running program will pause, return to the CCS command mode and print the message PAUSE. The computer will then type the line number at which the pause occurred and a >. Direct statements and CCS commands may then be executed. Execution of the program may be resumed at the point of interruption by typing CONTINUE.

8. STOP Statement

The form is

```
STOP
```

When a STOP statement is encountered, execution of the program is terminated. The user may enter statements for direct execution but the program cannot be continued.

9. END Statement

The form is

```
END
```

CCS FORTRAN IV programs and subprograms must end with an END statement. The END statement may not be labeled with a statement number; that is, it is not possible to transfer to this statement from other parts of the program.

D. INPUT/OUTPUT STATEMENTS

GENERAL

Input and output statements are, as their name implies, the statements used to transfer (communicate) data to and results from the computer. Input and output may be from the terminal or from data files stored on the disk. Terminal input and output always is symbolic and may be in free format (ACCEPT, DISPLAY) or in formatted (READ, WRITE, FORMAT) form.

The data files may be in either symbolic or binary form. Data files always must be opened before use and closed after use (OPEN, CLOSE). Symbolic files always are input and output in formatted form (READ, WRITE, FORMAT). Binary files always are input and output in a format free form (READ, WRITE).

FREE FORMAT TERMINAL INPUT AND OUTPUT - ACCEPT, DISPLAY

The simplest way to input and output data on the terminal is to use the free format input/output statements ACCEPT and DISPLAY which are unique to the Tymshare system. The form of the input command is as follows:

ACCEPT list

Example

```
ACCEPT A, B, (C(I),I=1,9)
```

Input List

The input list may include any legal FORTRAN IV variable and/or array name. As seen in the example above, an implied DO loop may be used to input part of an array. For example, the implied DO loop (C(I), I=1,9) will cause values for the variable C(1) through C(9) to be requested. An entire array which has been dimensioned previously in the program may be requested by its name. If, for example, B in the above statement had been dimensioned B(15), the statement ACCEPT B would request the array elements B(1) through B(15).

Input In Response To An ACCEPT Statement

When the ACCEPT statement is executed, the system will ring a bell and wait for the user to fill the variables and/or arrays specified. The user must then type in one value for each variable and/or array element specified in the variable list. The user may use a Carriage Return, a Line Feed, a comma, or a space bar to terminate each value input. The value supplied will assume the type (mode) of the variable or array in which it is stored. Unless otherwise specified in a TYPE statement, all variables and arrays beginning with I through N will store integer values; all other variables and arrays will store real values.

The DISPLAY command is used to output data to the terminal. The statement form of the DISPLAY command is as follows:

DISPLAY list

Example

```
DISPLAY A, B, (C(I), I = 1,9), 5*6 + 5, SQRT[ABS[-25]]
```

Variables, arrays, and expressions may be included in the output list. The DISPLAY command will cause the values stored in the variables and/or arrays specified to be printed on the terminal. Any expressions included in the list will be evaluated and their values returned.

Literal Text In The I/O Lists

An extension of the ACCEPT and DISPLAY statements allows the user to write literal text on the terminal either alone or in conjunction with variable values. The text desired may be enclosed in either single quotes, double quotes, or dollar signs as follows:

```
ACCEPT          'TEXT'
  or            "TEXT"
DISPLAY         $TEXT$
```

Examples

```
DISPLAY $THIS PROGRAM COMPUTES THE . . .$
ACCEPT "THE ORIGINAL VALUE IS = $", C
DISPLAY 'THE MAXIMUM CAPACITY = ', T, "TONS"
```

FORMATTED INPUT AND OUTPUT - READ, WRITE, FORMAT

Data also may be input and output in formatted form using the ASA Standard FORTRAN IV statements READ, WRITE, and FORMAT. The READ and WRITE statements are the executable input/output statements. The FORMAT statement is a non-executable reference statement which supplies certain information about the size and mode of the data values being read or written. The READ, WRITE, and FORMAT statements are used both for terminal input and output and for symbolic disk file input and output. The only difference between formatted terminal input/output and formatted symbolic disk file input/output is that symbolic disk files must be opened before use and closed after use. The general form of the input/output statements is:

READ
 or $\begin{pmatrix} \text{file} & \text{format} \\ \text{number} & \text{number} \end{pmatrix}$ list
WRITE

The format number refers to the label of the FORMAT statement used to read or write the values of the variables and/or arrays in the list.

The file number indicates the file from which the data is to be read or on which it is to be stored. If a disk file is being used, the file number specified is the one used in the OPEN statement. The file number in this case may be 2, 3, or 4. If the terminal is being used, the file number will be 0 or 1. The file numbers 0 and 1 are reserved for terminal input and output. The file number 0 specifies terminal input; the file number 1 specifies terminal output.

The input list may contain variables and arrays and implied DO loops. In addition to these, the output list also may contain expressions and constants (including Hollerith constants "ABC", 'ABC', ABC, 3HABC).

Examples

```
READ (3,7) A, I, (C(I), I = 1, 10)
WRITE (1,90) A, (B(R) R = 200,300,5), "5TG", SQRT[79]
```

FORMAT Statement

The FORMAT statement is a non-executable statement which specifies the size and type of the data values being read or written. The FORMAT statement consists of a set of field specifications which indicate how each value is to be read or written. The form of the FORMAT statement is:

format number FORMAT (field $spec_1$, field $spec_2$,...)

Example

```
90  FORMAT (I5,2X,F7.2,7(3(I4),2(F9.3)), A5)
```

Field Specifications

The field specification supplies certain information about the size and mode of the value being read or written. The information supplied by the field specification is used somewhat differently for input than for output.

During Input. The size specification is crucial since it indicates the number of characters to be read from the file. The mode of the field being read is occasionally completely ignored (for example, an exponential value may be read using a decimal field specification). The mode of the variable into which the value is being read overrides the mode specified in the FORMAT statement.

During Output. The type of the field specified always determines the mode of the value output. For example, an F field will always output the value as a decimal; an I field will always output the value as an integer. The size specification indicates the number of spaces that will be used to print the number. The decimal places will be supplied as indicated in the FORMAT statement.

NOTE: The field size is specified differently for the different types of fields. Careful attention should be paid to exactly how the field size is specified and what is included in the size specification.

Quick Reference Chart--Field Specifications

Field Type	Manner Specified	Usage
I	rIw	Integer field (123)
F	rFw.d	External fixed point decimal (1.23)
E	rEw.d	Floating point (1.E09)
J	rJi.d	Floating point (1.E09)
D	rDw.d	Double precision (1.D09)
G	rGw.d	Generalized (for E formats)
L	rLw	Logical (T or F)
A	rAw	Alphanumeric (JONES)
H	wHs	Hollerith (3HEND)
"	"s"	("END")
'	's'	('END')
$	s	(END)
X	wX	Spacing - spaces w times
T	Tw	Tab (spaces to column w)
P	fP	Scaling
/	/	Generates a Carriage Return
Z	Z	Suppresses a Carriage Return

Symbols

w - field width (entire number of characters required)
d - number of decimal digits
i - number of integer digits
s - string of characters
f - power of 10
r - repeat count

Numeric Fields

I The integer field is represented by the letter I. The general format for an integer field specification is Iw where w represents the maximum length of the number which can be input or output. If the number is negative, one extra digit should be specified for the negative sign.

Input

If the entire field is filled on input; for example, 123 read with I3, the number being read must be an integer (decimal points, E's, and D's will not be allowed in the field).

However, if the number is terminated by a comma before the entire field is filled, free form input is assumed and any type of number (such as 1.5, 2E07) may be input. In this case, all characters in the number will be counted. For example, -1E 07 has 6 characters so, if a comma terminator is used, a minimum field specification of I7 would be required. An I6 field would be filled completely.

Output

Any number output using an I field specification will be output as an integer. Any decimal digits will be truncated. If the field width w is larger than the number, leading blanks will be supplied. If it is smaller, an error diagnostic will be given.

F The external fixed point field (decimal) is represented by the letter F. The general form of the F field is Fw.d where w represents the entire width of the field which includes the plus or minus sign, the total number of characters in the field, the decimal point and any blanks, and the d indicates the number of decimal digits in the number.

Input

Any numeric constant (integer, real, double precision)[1] may be read into a variable or array element using an F format. The w indicates the number of characters to be read, and the d indicates the number of decimal positions. If, for example, the number 12345 were read using an F5.2 field specification, the number would be stored in the variable as 123.45. If the number being read has a decimal point in it, the d specification will be ignored. Thus, the number 123.4 read with the format F5.2 would be stored as 123.4.

Output

Numbers output using an F field will be truncated to d decimal digits. If the width w is larger than required, the number will be filled in with leading blanks. For example, the number -1.59764 would be output as _-1.597 using the field specification F7.3 and as _-1.597640 using a field specification F10.6. If insufficient width (w) is specified, an error message will occur.

E The floating point or exponential field appears in the following general form, Ew.d where w represents the entire width of the field including the plus or minus signs, the decimal point, all numeric characters, and the exponent; and d indicates the number of decimal digits.

D The double precision or D field is specified in exactly the same manner as the E field except that the exponent is written with a D instead of an E; that is, Dw.d.

Input

Any numeric constant may be input using the E and D format specifications.[1] The number of characters specified by the w will be read and d decimal digits assigned. For example, the number

+12345E+07 would be read, using the format E10.2, as 123.45E+07. If the number being read has a decimal point in it, the d specification will be ignored. Thus, -1.23D-05 would be read by D10.2 as -1.230D-05.

Output

Numbers output using an E or D format will be truncated to d decimal places. When the E field specification is used, the number will be output in E notation (1.5E+02). When the D field specification is used, the number will be output in D notation (1.5D+02).

J The J field also is used with floating point numbers. The general form of the J format, however, is Ji.d where i represents the number of integer digits and d represents the number of decimal digits.

Input

Any numeric constant may be input using a J field specification.[1] When the field is input, i integers will be read followed by d decimal digits. If the number is input in exponential form, the exponential part of the number (E±07) is not specified in the field specification. For example, the number 12345E+04 would be read by J3.2 as 123.45 E+04. If the number being read has a decimal point, the decimal position specified by d will be ignored. For example, the number 123.45E-7 would be read by J4.3 as 123.450E-07.

Output

Numbers output with a J format will contain i integer digits, d decimal digits, and the exponent. The decimal digits will be truncated if necessary. For example, the number 123.45 E+07 will be output as 123.4 E+07 by the format J4.1.

G The G, or generalized field, is a combination of the F and E fields. The general form is Gw.d where w represents the total field length and d represents the number of decimal digits.

Input

Same as E and D field specification. Any numeric constant may be input; w characters will be read.

Output

The G field specification will output the number in decimal or exponential form depending upon its size.

---

[1] A complex number may be read also. Complex numbers require a separate field specification for the real and for the imaginary parts of the number. Thus, the number 2.3, 1E07 could be read using the format statement FORMAT(F4.1,E6.0)

The number (n) will be output in F format (without an exponent) if it is within the range $0.1 < n < 10^d$, otherwise the number will be output in E format. For example, using the specification G12.3 the number 244. will be printed as 244. The number 2444. will be printed as .244E+04.

Scaling - P

A scaling factor may be used in a FORMAT statement to alter the normal form of the E, D, G, and F fields. The scaling notation is in the form fP where f indicates a power of ten by which scaling is to occur. Thus, 1P would indicate multiplication by ten; 2P multiplication by 100 (10^2); -1P would indicate multiplication by .1 (10^{-1}).

Scaling may be set and reset anywhere in the FORMAT statement. Once scaling is set it will be used throughout the program until it is reset. 0P (10 to the 0^{th} power, or 1) may be used to set the scaling back to normal.

```
FORMAT (1P,I4,F7.9,0P,F5.7,-2P,E7.6)
```

Input

When a number is input as a decimal in any of the above formats, the actual value of the number will be altered. If the number is input in exponential (1E07) or double precision (1D07) form, it will not be affected.

Value Input	Scaling Factor	Value Stored
109.3	1P	1093.
1.2E05	1P	1.2E05
111.3	-2P	1.113

Output

During output scaling is determined by the field specification used. If an F field specification is used, the actual value of the number will be altered. When E and D field specifications are used, the decimal point and the exponent will be realigned, but the actual value of the number will not be altered. When a G field specification is used, the decimal point and exponent will be realigned if the number is output in exponential form. If the number is output in decimal form, it will not be affected by the scaling.

Format	Normal Output	Scaled Format	Scaled Output
F4.0	15.	1PF4.0	150.
E12.5	.15000E 02	1PE12.5	1.5000E 01
D12.5	.15000D 02	-1PD12.5	.01500D 03
G3.1	1.5	-1PG3.1	1.5

Non-Numeric Fields

L

The logical field L is used with the logical values T(.TRUE.) and F(.FALSE.). It appears in the form Lw where w represents the number of characters in the logical field. Whenever a logical value is read, the first character read must be either a T or an F or an error message occurs. If a T is encountered, the logical variable will be assigned a value of .TRUE.; if an F is encountered, the logical variable will be assigned a value of .FALSE. . When a logical value is written, a T or an F always will be output. If the output field size is greater than one, leading blanks will be supplied automatically to fill the field.

A

The alphanumeric or A field appears in the general form Aw where w is the number of characters in the field. Any group of characters may be stored in or read from a variable using the A format. For example, character set J.J.JONES, 23. could be stored in a variable which is read or written using the field specification A14.

NOTE: Do not read or write numeric values using an A format. Use one of the numeric formats.

H (Hollerith Field)

The Hollerith field which appears in the general form wH causes the number of characters specified by w, which immediately follow the H, to be written on a file or the terminal during output. During input the number of characters specified (w) will be skipped. For example, the format FORMAT (8HTYMSHARE) would print TYMSHARE during output and skip 8 characters during input.

",',$

A Hollerith field also may be specified by enclosing it in double or single quotes or dollar signs. The general form is "s", 's', or s where s indicates a string of characters. The string will be printed on the terminal or output to a file during output. During input the number of characters specified in the string will be skipped. For example, FORMAT ("THE ", 'BIG ', DOG) would during input, skip the first 11 characters, but during output would print THE BIG DOG.

Spacing

X

The X specification, which appears in the form wX, causes w blanks to be printed during output or w columns to be skipped during input. For example, 3X will leave three blank spaces during output or not read 3 characters during input.

T

The T specification appears in the form Tw. Using the first column as a reference point, it spaces to the column specified by w. Thus, T40 will start reading or writing the next field in column 40. The T specification may be used to backspace during output but not on input. For example, the statements

```
WRITE (1,1)A, B
FORMAT (T50,I6,T10,F5.2)
```

would output the value stored in A starting in column 50 and then backspace to column 10 and output the value stored in B.

Generating And Inhibiting A Carriage Return - /,Z

A slash (/) is used in a FORMAT statement to generate a Carriage Return, which causes both reading or writing to skip to the next record. Multiple slashes may be used to skip more than one record when reading a file, or to generate blank records when writing a file. This allows the user to include the fields specifications for more than one record in a single format statement.

If A = 111, B = 2.135E06, and C = 5.794, the statement

```
WRITE (1,100) A,B,C, "ABCF"
100 FORMAT (I3,3X,E10.3/,F5.3,5X,A4)
```

will output the following:

```
111       .213E+07        (record 1)
5.794       ABCF          (record 2)
```

NOTE: When a slash is used between two field specifications, a comma need not be used to separate the specifications; for example, FORMAT (I5,F6.3/G8.0)

A Z may be used at the end of a FORMAT statement to suppress the Carriage Return that is normally generated each time the FORMAT statement is scanned. This allows writing on or reading from a single record using more than one FORMAT statement. (This feature also is very useful in documenting formatted terminal input.)

Example

The statements below

```
WRITE (1,100)
100 FORMAT ('VELOCITY=',Z)
READ (0,200) V
200 FORMAT (F6.2)
```

will print

```
VELOCITY =
```

and then wait on the same line for the user to input the value of V. (The Carriage Return that would normally be generated after 100 FORMAT was scanned was suppressed by the Z.)

Repeating A Field Specification

If the same field specification is to be used a number of times, a repeat count may be used. The repeat count appears immediately preceding the field specification and specifies the number of times the field specification is to be read or written. For example, the format statement

```
FORMAT (3I4,7F9.6)
```

will read or write the first three values as four digit integers and the last seven values as nine place fixed point values with six decimal places.

Parentheses may be used to indicate the repetition of a series of field specifications. The format statement

```
FORMAT (2(2I3), 3(I4,I5))
```

will read or write the first four values as three digit integers (2(2I3)); it will then repeat the specifications (four digit integer, five digit integer) three times.

Repeating FORMAT Statements

A FORMAT statement will be rescanned automatically whenever the number of items specified in the input or output list exceeds the number of fields specified in the FORMAT statement. Each time the FORMAT statement is scanned a Carriage Return will be generated and reading or writing will be continued on the next line (record).

Data Field/Field Specification Compatibility

The term data field refers to a single value or string of characters. Every data field must be read or written using the corresponding field specification in the FORMAT statement and must be stored in an array element or variable.

An input data field may appear in one of three forms:

1. Field filled--no decimal point. In this case the data appears as a solid string of characters not including decimal points. (NOTE: Blanks are treated as zeros in numeric fields.) The number of characters read in the field is the same as the field width specified in the field specification. A data field which fills the field specification entirely is assigned its value according to the field specification.

Data Field	Field Specification	Value Assigned
12345	F5.2	123.45
123E04	E6.1	12.3E04
123	I3	123
ABCDE	A5	ABCDE
12345	J1.4	1.234
TRY	L3	T (true)

2. Field filled--with decimal points. In this case the data also appears as a solid string of characters except that a decimal point may be included in a numeric value. The decimal position specified in the input field overrides any decimal position specification given in the field specification. The number of characters specified in the field specification is the number of characters that will be read. (The decimal point is counted as a character.)

Data Field	Field Specification	Value Assigned
1.2345	F6.2	1.23
123.E04	E7.2	123.E04
.123	J1.4	0.123

3. Field not filled--terminated with comma. A comma may be used to terminate the input before the entire field specified has been filled. Whenever a comma is encountered in a data field, reading is discontinued and blanks are supplied to fill the field. If a blank field is terminated by a comma, a value of 0.0 is assigned. A field terminated by a comma will be read in free-format form; that is, the number will be stored exactly as it appears; the field specification will be ignored entirely. (Remember--the number of characters in the field must be fewer than the number specified as the field width.) Using the free-format comma field terminator relieves the user of the time-consuming task of filling all of the fields, thus minimizing the time required to create the file and also the space required to store the file. One exception exists: Alphanumeric (A) fields are not terminated by a comma since a comma is considered a valid character in an A field. However, an alphanumeric field may be terminated by a Carriage Return or a Line Feed. (NOTE: A Carriage Return or Line Feed also terminates a record.)

Data Field	Field Specification	Value Assigned
33.1,	F3.3	33.1
44.2,	I5	44.2 (if read into a real variable)
44,	F7.2	44.
,	F5.3	0.0

If the data field and the field specification are not compatible; that is, not the same size and in some cases not the same type (mode),[1] the data may be read or written improperly or an error message returned. If for example, the input data field is shorter than the one specified in the field specification using filled field data, only part of the data value will be read. The rest of the data value will be read as part of the second value. Conversely, if the input data field is longer than the field width specified, only the first portion of the data value (w characters) will be read into the value; the rest of the field will be treated as part of the next field.

Example

A should be 12.34; B should be 5.12. When the field width was counted the decimal point was forgotten.

The file appears as: 12.345.12 and is read by:

```
    READ(3,1) A,B
  1 FORMAT(F4.2,F3.2)
```

As:

A = 12.3 B = 45.

If the output data field is shorter than indicated in the field specification, blanks are supplied to fill the field. If the output data field is larger than the field width specified, decimal digits in the data value will be truncated whenever possible to reduce the data value to the field width specified. If the value cannot be truncated; for example, if the data field is 12345 and the field specification is I4, program execution will be interrupted and an error message returned.

Data Records

A data file is composed of data records. A data record consists of any number of concatenated data fields which are terminated by a Carriage Return or a Line Feed. A data record therefore normally appears on a physical line. A data record may be continued on the next line by including an exclamation point (!) just before the Carriage Return or Line Feed. When an exclamation point is encountered, reading is transferred immediately to the beginning of the next line ignoring the rest of the current line; namely, the Carriage Return or Line Feed. By using the exclamation point, a data record of any length can be written.[2]

---

[1] If the field is entirely filled, an I field specification may be used only with integer data. For a complete list of such restrictions, see the section on field specifications.

[2] On output, a record of any length may be created by suppressing the Carriage Return with a Z in the format.

Records might appear as follows:

```
Record 1    12345678,910,111.11167249.7
Record 2    1795.3,7,11,,65411,3!
            213579.6
```

Data Record - Format Compatibility

Normally, every data record is read using a separate FORMAT statement or by rescanning an entire FORMAT statement with each data value (data field) being read by its corresponding field specification. The Carriage Return (or Line Feed) at the end of each record corresponds to the Carriage Return supplied each time a FORMAT statement is scanned.

Example

The data file:

```
1234.56789AAC
7965.3477,8RTF
9773.996RMT
```

will be read by:

```
      READ (3,9) N1,A1,B1,R1
 9    FORMAT (I2, F5.2, F3.1, A3)
      READ (3,9) N2,A2,B2,R2
      READ (3,10) X,Y,Z,L
10    FORMAT (I3, 4X, I1, A2, L1)
```

as:

```
N1 = 12    A1 = 34.56    B1 = 78.9    R1 = AAC
N2 = 79    A2 = 65.34    B2 = 77.0    R2 = 8RT
 X = 977    Y = 6         Z = RM       L = T (true)
```

It is possible however, to read or write a number of data records using a single FORMAT statement by specifying a slash (/) in the FORMAT Statement at the end of the field specifications for each record. Remember, a slash (/) generates a Carriage Return which will cause a skip to the beginning of the next record. Multiple slashes may be used in the FORMAT Statement to skip records on input or generate blank records on output.[1]

Example

```
The data file      12345,6789
                   12345678
will be read by    READ (1,7) A,B,I,D,E
                 7 FORMAT (F4.2,F5.2,I2/F5.2,J3.4)
```

[1]A slash may be used in place of the comma to separate the field specifications.

as A = 12.34 B = 5.00 I = 67 D = 12.34 E = 5.678

It is possible also to read or write on a single record using more than one FORMAT statement by including a Z at the end of the FORMAT. The Z suppresses the Carriage Return that is normally issued each time a FORMAT statement is scanned.

The entire record or file need not be read. For example, the following data file:

```
JONES,J.J. 29546793
SMITH,R.V.113476995
```

will be read by:

```
    READ (3,10) (A(I),N(I) I=1,M)
10  FORMAT (A10,I5)
```

as:

```
A(1) = JONES,J.J.       M(1) =  2954
A(2) = SMITH,R.V.       M(2) = 11347
```

If the format indicates that more data values are to be read from a record than actually exist on that record, zero values will be supplied at the end to fill the record.

Example

The four record data file

```
1111111↵
2222222↵
3333333↵
4444444↵
```

will be read by

```
    READ (3,100) A,B,C,D
100 FORMAT (4F5.2)
```

All values read from the same record since no / indicated; zeros supplied to fill the field.

as

A = 111.11 B = 0.11 C = 0.0 D = 0.0

Further Examples

The format statements

```
    READ (3,100) A,B,C,D
100 FORMAT (4(F5.2/))
```

and

```
    READ (3,100) A,B,C,D
100 FORMAT (F5.2)
```

will both read one value from each record as follows:

A = 111.11 B = 222.22 C = 333.33 D = 444.44

If a slash were included in the second statement above,

```
    READ (3,100) A,B,C,D
100 FORMAT (F5.2/)
```

every other record would be read, A would be A = 111.11, B would be B = 333.33, an end of file would be encountered, and the error message UNEXPECTED END OF FILE printed. Double slashes in the first statement; for example, FORMAT (4(F5.2//)) would have produced the same result.

The file

```
16,,5643,,
```

will be read by the format statement

```
    READ (3,60) A,B,C,D
 60 FORMAT (F5.2)
```

as

A = 0.16 B = 0.0 C = 56.43 D = 0.0

If a blank field is terminated by a comma, a value of zero is assigned.

Constructing A Symbolic Data File

A symbolic data file may be constructed in a number of ways:

1. It may be read from paper tape.
2. It may be constructed using the EDITOR commands READ, WRITE and APPEND.
3. It may be constructed using the EXECUTIVE command COPY.
4. It may be created by a FORTRAN IV program.

The data file may contain any number of data records, each terminated by a Carriage Return or a Line Feed. Each data record may contain any number of data fields. The data fields may be filled completely or terminated by commas. The data fields must be compatible with the field specifications used to read them. The data records must be compatible with the FORMAT statements used. Enough data must exist on the file to fill all of the variables and/or array elements specified according to the formats specified or an End of Data error diagnostic will be given and execution of the program halted.

Terminal Input In Response To A READ Command

Data input from the terminal in response to a READ command is treated as if it were a symbolic data file. When the READ command is executed, the system will ring a bell and wait for the user to input one piece of data for each variable or array element in the list. Each data value should then be input using one of the three data field forms. The data must follow the field specifications given in the FORMAT statement.

DISK FILE INPUT/OUTPUT - OPEN, CLOSE

Disk file input/output may be either symbolic or binary. Symbolic disk file input/output is read and written in the formatted form explained in the preceding section. Binary input/output is read and written in a format free form explained later in this section. Unlike terminal input/output, disk files always must be opened before use and closed after use.

OPEN And CLOSE Statements

To read or write on a disk file, the file first must be opened for input or output and its form, symbolic or binary, specified. The form of the OPEN statement is as follows:

$$\text{OPEN}\ \left[\begin{matrix}\text{file}\\ \text{number}\end{matrix},/\begin{matrix}\text{file}\\ \text{name}\end{matrix}/,\ \begin{matrix}\text{INPUT}\\ \text{or}\\ \text{OUTPUT}\end{matrix},\ \begin{matrix}\text{SYMBOLIC}\\ \text{or}\\ \text{BINARY}\end{matrix}\right]$$

File numbers 2, 3, and 4 are used for disk file input and output. Any file name may be used, although it is suggested that the file name be short (four or five characters).[1] A maximum of three files are allowed open at one time.

After the last input or output operation has been performed with a disk file, the disk file should be closed. The form of the CLOSE statement is:

CLOSE (file number)

After a file is closed it must be reopened again before it can be reused. Whenever a file is closed and reopened, reading or writing starts automatically at the beginning of the file. There is no way to continue reading or writing on a file once it has been closed without rereading or rewriting to the point at which the user wishes to continue.

NOTE: File numbers 0 and 1 are reserved for terminal I/O and are never used in an OPEN statement. Note, however, that when a COMMANDS FROM file is used, it is closed using the command CLOSE(0). (See Commands From Files, page 00.)

Binary Disk File Input/Output

Binary files also may be read and written by a FORTRAN IV program. These files have the disadvantage of not being readable in the EXECUTIVE or in EDITOR since they are written in a binary code, but they have the advantage of requiring less storage space than a symbolic file. Binary files must be

---

[1] Another user's data file may be opened for input if it has been declared public or contains an @ in the file name as follows:

OPEN [#,"(Account Number User Name)/@FILE/", INPUT, SYMBOLIC or BINARY]

Example

```
OPEN [3,"(A3R)/@DATA/",INPUT,SYMBOLIC]
```

opened for binary input or output in the OPEN statement. They are read and written using a format-free form of the READ and WRITE statements as follows:

$$\begin{matrix}\text{READ} \\ \text{or} \\ \text{WRITE}\end{matrix} \left(\begin{matrix}\text{file} \\ \text{number}\end{matrix}\right) \text{ list.}$$

Example

```
OPEN [3,/BIN/,INPUT,BINARY]
READ (3) A, B, C, (R(I) I=1,9)
CLOSE (3)
```

E. DECLARATION STATEMENTS

It is essential in CCS FORTRAN IV, as in other FORTRAN IV compilers, that specific areas internal to the computer be reserved and that certain data relevant to program operation be supplied by the program. This information and data is furnished to the compiler in the form of "non-executable" statements called Declaration Statements.

Two of the declaration statements, END and FORMAT, have already been introduced. The FORMAT declaration is the only declaration statement that ever appears with a label. Nine declaration statements will be introduced in this section: Comment, DIMENSION, COMMON, Unlabeled DATA, and the Type Declarations (INTEGER, REAL, COMPLEX, LOGICAL, DOUBLE PRECISION). Four other declaration statements will be introduced in the following sections: SEGMENT, FUNCTION, SUBPROGRAM, and RETURN.

COMMENT DECLARATION

A comment consists of a string of characters commonly used to supply information (documentation) on the program. Comments are ignored entirely when the program is executed. A comment may appear anywhere in the program except as the last statement (END must be the last statement). A comment is indicated by placing either a C: or an * at the beginning of the line as follows:

```
C:THIS IS A COMMENT
*COMMENTS COMMONLY ARE USED FOR PROGRAM DOCUMENTATION
```

DATA STATEMENTS

Occasionally it is desirable to have certain data values reside within the program rather than use input or assignment statements. The DATA statement allows the user to store his data in the program. A DATA statement has two general forms as follows:

DATA $Variable_1$/$Value_1$/,$Variable_2$/$Value_2$/

DATA $Variable_1$,$Variable_2$,$Variable_3$/$Value_1$,$Value_2$,$Value_3$/

For example, both of the following DATA statements would set A = 1.1, B = 2E07, C = T (true), and I = 110.

```
DATA A/1.1/,B/2E07/,C/T/,I/110/
DATA A,B,C,I/1.1,2E07,T,110/
```

Only constants and logical values (T and F) may be used as values in a DATA statement. An implied DO loop may be used with the second form of the DATA statement to assign values to an array as follows:

```
DATA (Z(I), I=1,5)/1,2,3,4,5/
```

If the same data value is to be repeated, it may be specified by placing the number of repetitions desired, an asterisk (*), and the data value inside of the slashes as follows. The DATA statement

```
DATA (Z(I),I=1,5)/5*2/
```

will assign a value of two to all elements of the array A(1) through A(5). If the number of values does not match the number of variables, the value list will be rescanned from left to right until all of the variables are filled. The DATA statement

```
DATA (Z(I),I=1,5)/1,2,3/
```

will assign a value of one to Z(1) and Z(4), a value of two to Z(2) and Z(5), and a value of three to Z(3).

Data statements may appear with or without a statement label as:

```
    DATA A/5/      unlabeled
  2 DATA B/7/      labeled
```

Labeled and unlabeled data statements are treated quite differently and it is important that the user understand this difference. Unlabeled data statements are used to initialize variables, and are executed only once; namely, just before the first executable statement in the program. Labeled data statements technically are not declaration statements since they are executable statements. They are executed according to their position in the program, they are executed every time that they are encountered, and may be used to initialize and reinitialize variable values.

Example

```
>ENTER 10(10)

 10.          1 CONTINUE
 20.            DATA A/1/
 30.          2 DATA B/2/
 40.            DISPLAY A,B
 50.            A=3
 60.            B=4
 70.            GO TO 1
 80.            END
 90.    (D^C)

>RUN
   1.    2.
   3.    2.
   3.    2.
   3.    2.
   3.    2.
   3.    2.
   3.    2.
   3.    2.
   3.
```

DIMENSION DECLARATION

All arrays used in FORTRAN IV must be dimensioned to reserve sufficient space in the computer memory for all elements of the array. Dimension statements identify all array variables in the program by name and indicate how many memory locations are to be assigned to each array variable. The general form for a one dimensional array is as follows:

```
DIMENSION array name (first element:last element)
```

Any legal variable name may be used as an array name. However, if a variable is used as an array name, it should not be used as a variable name. The boundaries of the array appear in parentheses. The lower boundary or first element is presented first; the upper boundary or last element of the array is presented last. The array elements must be specified by integer constants in the main program. In a subprogram they may be specified by variables. The elements may be positive or negative integers. For example, the declaration

```
DIMENSION A(-4:6)
```

would reserve space for the eleven element array A(-4) through A(6). If the first element of an array is one, only the last element need be listed.

For example, the declaration

```
DIMENSION MIN(25)
```

is equivalent to the declaration

```
DIMENSION MIN(1:25)
```

Multidimensional arrays use an extended version of the DIMENSION declaration as follows:

$$\text{DIMENSION } \begin{matrix}\text{array}\\ \text{name}\end{matrix} \left(\begin{matrix}\text{first}\\ \text{element}_1\end{matrix} : \begin{matrix}\text{last}\\ \text{element}_1\end{matrix}, \begin{matrix}\text{first}\\ \text{element}_2\end{matrix} : \begin{matrix}\text{last}\\ \text{element}_2\end{matrix}, \ldots\right)$$

For example, the dimension declaration

```
DIMENSION R(0:6,-4:4)
```

would set up an array with the range R(0,-4) through R(6,4).

Space is reserved for multidimensional arrays as though they were linear strings. The elements are stored in the string column by column varying the first dimension first, the second second, etc. For example, the array A(M,N) would be stored as follows:

$$A_{1,1}, A_{2,1}, A_{3,1}, \ldots A_{M,1}, A_{1,2}, A_{2,2}, A_{3,2}, \ldots A_{M,2}, \ldots A_{M,N}$$

An array also may be dimensioned in a Type Declaration statement using implied dimensioning. Implied dimensioning; for example, COMPLEX A(10,5), will be covered in the section on Type Declaration.

If an array is dimensioned in the main program, there is no need to redimension it in a subprogram if it is passed to the subprogram as a parameter. If a COMMON declaration is used to pass the array however, the array must be redimensioned in the subprogram.

COMMON DECLARATION

COMMON declarations are used to assign data to an area of "common storage" which may be referred to by the main program and all subprograms (subroutines, functions, and segments) in which a COMMON declaration appears. Unless a variable is specifically declared to be COMMON it will exist (have a particular value) only in the main program or subprogram in which it was given the value. The COMMON declaration has the following form:

```
COMMON variable list
```

Example

```
COMMON A,B,I
```

The variable list consists of the names of single variables or arrays which are assigned to common storage in the order in which they are listed. The COMMON declaration is position oriented; that is, the variable name itself is not important (except that it specifies the mode of the data), it is the variable position in the list that must be considered. For this reason a common value need not have the same variable name in the subprogram that it does in the main program. The variables must, however, have the same mode (integer, real, etc.).[1]

Example

```
COMMON A,B,C      Main Program
COMMON X,Y,Z      Subprogram
```

In the above situation A has the same value as X, B the same as Y, and C the same as Z.

NOTE: If an array is passed to a subprogram using just the array name, the array must be redimensioned in the subprogram.

```
DIMENSION A(10,10) }
COMMON A           } Main Program

COMMON C           }
DIMENSION C(10,10) } Subprogram
```

If the array dimensions are included in the COMMON statement, the array is not redimensioned in the subprogram.

```
DIMENSION A(10,10) }
COMMON A(10,10)    } Main Program

COMMON C(10,10)      Subprogram
```

---

[1] One exception exists to this rule. REAL and INTEGER modes may be interchanged. This is possible because REAL and INTEGER values both occupy two words of storage. Since the COMMON statement is position (word length) oriented, a variable which is declared REAL in the main program and INTEGER in the subprogram will still be reading the correct number of words and therefore maintain the correct position orientation.

TYPE DECLARATIONS

Type Declaration is used to override the implied mode of a variable or function name. Unless the mode of the variable or function name has been changed explicitly in a type statement, all variables and functions beginning with the letters I, J, K, L, M, or N are typed as integers and all other variables and function names are typed as real. The general form of the Type Declaration Statement is as follows:

```
Type of Variables variable list
```

The type specified may be INTEGER, REAL, DOUBLE PRECISION, COMPLEX, or LOGICAL.

Examples

```
INTEGER A,POUND,GAL,R
REAL I,MIN,MAX
DOUBLE PRECISION ART,HAR,STD,MEAN,X
COMPLEX B,J,T,COM
LOGICAL FIR,SEC,TR,N
```

The variables in the list following the Type Declaration Statement are all declared to be of that mode; thus, A,POUND,GAL,R store integer values; B,J,T,COM store complex values, etc.

A program may have as many Type Declaration Statements as are required. All of the Type Declarations are executed before the first "executable" step in the program; therefore, a type declaration may not be used to change the mode of a value while the program is being run. The mode of a value may be changed during program execution by assigning the value to a new variable (a value assumes the mode of the variable in which it is stored), or by using one of the intrinsic functions specifically designed to convert a variable from one mode to another.

Dimensioning With Type Declaration Statements

The Type Declaration Statement also may be used as an implied dimension statement. Variables that are arrays may be subscripted in the Type Declaration to indicate the desired dimensions of the arrays, thus eliminating the need for a DIMENSION statement. For example, the following DIMENSION and Type Declaration statements

```
REAL      MIN,MAX
DIMENSION MIN(16,16),MAX(20,20)
```

may be replaced by a single Type statement using implied dimensioning as follows:

```
REAL  MIN(16,16),MAX(20,20)
```

F. SUBPROGRAMS - FUNCTIONS AND SUBROUTINES

A subprogram consists of a group of one or more statements which are stored together outside the main program and which may be called by a name assigned to the group. There are two general categories of subprograms; the function subprogram and the subroutine subprogram.

A function subprogram is designed to return a single result and is called merely by the appearance of the function name with the appropriate arguments. Three categories of function subprograms are covered below: the Statement Function, Library Functions, and the Function Subprogram.

A subroutine subprogram is not designed to return a specific result. A subroutine must be called explicitly by a CALL statement and may or may not return values.

Two separate steps are necessary if a subprogram is to be used:

1. The function or subroutine must be defined unless it is a library function (which is defined internally).
2. The function or subroutine must be called.

STATEMENT FUNCTIONS

Statement functions are used when a single expression is to be performed repeatedly in the main program or one of the subprograms.

Defining A Statement Function

Statement functions are defined with a single statement as follows.

function name [dummy arguments] = expression.

Example

```
FUNCT[A,B] = A**2 + B*2 - C
```

Any legal variable name may be used as a function name. The function will be of the same mode as this variable. The dummy arguments may be simple variables or array names (they may not be constants, subscripted variables, nor expressions). The dummy arguments are replaced at the time of execution by the actual arguments, which are supplied when the function is called. Every function must have at least one dummy argument. The dummy arguments are not considered to have mode. Mode is assumed only after the actual argument has replaced the dummy argument.

The expression defines the computations which are to be performed whenever the function is called. In addition to the variables appearing as dummy arguments, the expression also may contain variables from the program or subprogram in which the function is defined and called. Note however that statement functions are not recursive; that is, the function name may never appear in the

expression. For example, the expression FUN[A,B] = A+B+FUN[3.1,4.1] is not allowed.

Calling A Statement Function

Statement functions may be called only in the program or subprogram (function, subroutine or segment) in which they are defined. The function is called by the appearance of the function name and the argument list (in brackets). A function may be used anywhere that an arithmetic expression may be used, such as in a replacement statement (A = FUNT[3.1,5.9] + 9*X↑2), or in a DISPLAY statement (DISPLAY FUNT[5,6]). The appearance of the function with the actual arguments in brackets causes the computations specified in the expression to be performed. The resulting value then replaces the function reference in the arithmetic expression, DISPLAY statement, etc. The value returned will assume the mode of the function. NOTE: Unless specified in a Type statement, all function names beginning with I through N will return integer values, all others will return real values. The actual arguments used in calling a function may be constants, subscripted or non-subscripted variables, array names, or arithmetic expressions.

Example

```
>ENTER 10(10)
10 C: MAIN PROGRAM
20 SROOT [A,B] = (A**2+B**2)**(1/2)
30 A = 3.0
40 ANS = SROOT [A,4]
50 DISPLAY 'SQUARE ROOT =', ANS
60 END
70 D^C
>RUN
```

The function was defined in statement 20. The function was called in statement 40. The function was not given a specific type, therefore the real mode was assumed.

LIBRARY FUNCTIONS

Many functions which are commonly used are stored permanently on the system as library functions. The library includes a variety of functions which may be used for computing trigonometric functions (angles must be specified in radians), comparing variable lists, converting modes of variables within an expression, computing logarithmic functions, and so forth.

The library functions may be used simply by naming the function and placing the argument list in brackets as follows:

```
function name  [arguments]
```

Example

SIN [2.3145]

The library functions are treated as expressions and may be used in replacement statements (A = SIN [1.596], in DISPLAY statements (DISPLAY EXP [1.957]), etc. The following list includes all of the CCS FORTRAN IV functions in the library. The list specifies the form of the function, the mathematical expression represented, the number of arguments required, the mode of the arguments, and the mode of the function.

Function	Statement Form	Expression	No. Arg's	Mode of Argument	Mode of Function
Arctangent	ATAN[arg]	arctan (a)	1	Real	Real
	DATAN[arg]		1	Double	Double
	ATAN2[arg_1,arg_2]		2	Real	Real
	DATAN2[arg_1,arg_2]	arctan (X,Y)	2	Double	Double
Cosine	COS[arg]	cos (a)	1	Real	Real
	DCOS[arg]		1	Double	Double
	CCOS[arg]		1	Complex	Complex
Hyperbolic Cosine	COSH[arg]	cosh (a)	1	Real	Real
Hyperbolic Sine	SINH[arg]	sinh (a)	1	Real	Real
Hyperbolic Tangent	TANH[arg]	tanh (a)	1	Real	Real
Sine	SIN[arg]	sin (a)	1	Real	Real
	DSIN[arg]		1	Double	Double
	CSIN[arg]		1	Complex	Complex
Exponential	EXP[arg]	e^a	1	Real	Real
	DEXP[arg]		1	Double	Double
	CEXP[arg]		1	Complex	Complex
Natural Logarithm	ALOG[arg]		1	Real	Real
	DLOG[arg]	ln (a)	1	Double	Double
	CLOG[arg]		1	Complex	Complex
Common Logarithm	ALOG10[arg]	$\log_{10}(a)$	1	Real	Real
	DLOG10[arg]		1	Double	Double
Square Root	SQRT[arg]		1	Real	Real
	DSQRT[arg]	$\sqrt{a}$	1	Double	Double
	CSQRT[arg]		1	Complex	Complex
Absolute Value	ABS[arg]		1	Real	Real
	IABS[arg]	$\|a\|$	1	Integer	Integer
	DABS[arg]		1	Double	Double
	CABS[arg]	$\sqrt{(a^2 + b^2)}$ for a + bi	1	Complex	Real
Maximum Value	AMAX0		2+	Integer	Real
	AMAX1		2+	Real	Real
	MAX0	Max(a_1,a_2..)	2+	Integer	Integer
	MAX1		2+	Real	Integer
	DMAX1		2+	Double	Double

Function	Statement Form	Expression	No. Arg's	Mode of Argument	Mode of Function
Minimum Value	AMIN0	$Min(a_1,a_2..)$	2+	Integer	Real
	AMIN1		2+	Real	Real
	MIN0		2+	Integer	Integer
	MIN1		2+	Real	Integer
	DMIN1		2+	Double	Double
Remaindering (Modular arithmetic)	AMOD[arg_1,arg_2]	a_1(mod a_2)	2	Real	Real
	DMOD[$arg_1$$arg_2$]		2	Double	Double
	MOD[arg_1,arg_2]		2	Integer	Integer
Positive Difference	DIM[arg_1,arg_2]	a_1-Min(a_1,a_2)	2	Real	Real
	IDIM[arg_1,arg_2]		2	Integer	Integer
Truncation	AINT		1	Real	Real
	INT		1	Real	Integer
	IDINT		1	Double	Integer
Float	FLOAT[arg]	Conversion Integer to Real	1	Integer	Real
Fix	IFIX[arg]	Conversion Real to Integer	1	Real	Integer
Transfer of Sign	SIGN[arg_1,arg_2]	Sign a_2 times $\|a_1\|$	2	Real	Real
	ISIGN[arg_1,arg_2]		2	Integer	Integer
	DSIGN[arg_1,arg_2]		2	Double	Double
Most Significant Part of Double Precision Argument	SNGL[arg]		1	Double	Real
Real Part of Complex Argument	REAL[arg]	a for a + bi	1	Complex	Real
Imaginary Part of Complex Argument	AIMAG[arg]	b for a + bi	1	Complex	Real
Single Precision Argument in Double Precision Form	DBLE[arg]		1	Real	Double
Two Real Arg's in Complex Form	CMPLX[arg_1,arg_2]	$a_1 + a_2 i$	2	Real	Complex
Obtain Conjugate of a Complex Argument	CONJG[arg]		1	Complex	Complex

FUNCTION SUBPROGRAMS

Most useful functions cannot be defined by a single expression and therefore must be stored in a function subprogram. A function subprogram may contain any set of computations desired, but is designed to return one specific value, although other values may be returned. The function subprogram possesses a certain degree of independence and often may be debugged and run separately from the main

program. A function subprogram may call any subprogram (function, subroutine, or segment) including itself, and thus may be used recursively.

Defining A Function Subprogram

A function subprogram always must have as its first statement a FUNCTION declaration and as its last statement an END declaration. Also, it must contain at least one RETURN statement, at least one dummy argument, and one value must be assigned to a variable with the same name as the function.

The FUNCTION declaration tells the system that it is now entering a function subprogram, supplies the name of the function subprogram, and lists the dummy arguments. The general form of the declaration is as follows:

```
FUNCTION   function name    [dummy arguments]
   :
   :
RETURN
   :
   :
 END
```

Any variable or array name may be used as a dummy argument. These arguments are merely place holders and will be replaced by the actual arguments specified when the function is called. At least one dummy argument must be specified, although more may be specified if desired. (Dummy arguments are covered more fully at the end of this section.)

Within the function subprogram, the value that is to be returned must be stored in a variable that has the same name as the function. The value stored in this variable will be returned to the place at which the function was called when a RETURN statement is executed.

Example Of A Function Subprogram

```
FUNCTION HYP [A,B]
IF ((A .EQ. 0) .OR. (B .EQ. 0)) GO TO 100
HYP = SQRT[A↑2 + B↑2]
RETURN
100 DISPLAY "ONE OF THE SIDES IS 0"
RETURN
END
```

Type Specification Of A Function Subprogram

The mode of the variable used as the function name determines the mode of the value returned. The implied mode will be assumed unless a specific type is specified using a Type Declaration Statement in the calling program or in the FUNCTION statement. If the mode of the function subprogram is to be specified in the FUNCTION declaration, the following general form is used:

Type of Variable FUNCTION function name [argument list]

The mode specified may be INTEGER, REAL, COMPLEX, LOGICAL, or DOUBLE PRECISION.

Example

```
REAL FUNCTION INT [A,B,I,R]
INT = A+B*I/R
RETURN
END
```

In the above example the value returned, INT, will be in the real mode. Had the type not been specified the integer mode would have been assumed.

Calling A Function Subprogram

A function subprogram is called automatically whenever its name appears followed by the actual arguments required. When the function is called, control will be transferred to the statement following the FUNCTION declaration. The function subprogram will be executed step by step until a RETURN statement is executed. Then the value presently stored in the variable with the same name as the function will be returned and execution of the main program continued.

A function call is treated as an arithmetic expression and may appear anywhere that an arithmetic expression is legal. The actual arguments may be specified as constants, predefined subscripted or non-subscripted variables, array names, or arithmetic expressions.

For example, the function subprogram HYP defined earlier might be called in a replacement statement.

```
R = 5.1
HY = HYP [R,3.2]
```

NOTE: All variable values are local to the subprogram in which they are defined. If the same variable value is to be used in a main and subprogram it must be "shared". Values, other than the "function value" which is automatically returned to the main program, may be "shared" by the main and subprograms in one of two ways. They may be passed either as parameters in the argument list or declared in COMMON using a COMMON declaration in both the main and subprogram.

Example

The function computes the mean of a series of numbers. The number of values in the series is given in the main program.

```
 10        C: MAIN PROGRAM
 20        DIMENSION R(200)
 30        ACCEPT R                                  Entire array accepted.
 40        RM = MEAN [R,200]                         Function subprogram called.
 50        DISPLAY $MEAN = $,RM
 60        END
 70        C: FUNCTION DEFINED
 80        REAL FUNCTION MEAN [S,N]
 90        Z = 0
100        DO 100 I = 1,N
110        Z = Z+S(I)
120   100  CONTINUE
130        MEAN = Z/N
140        RETURN
150        END
```

Following is an example of a recursive function subprogram (a subprogram which calls itself). The function computes the factorial of any number N.

```
 50        R = FACTORIAL [7]
 60        END
 70        FUNCTION FACTORIAL [N]
 80        IF (N.EQ.0) GO TO 5
 90        FACTORIAL = N*FACTORIAL [N-1]
100        RETURN
110     5  FACTORIAL = 1
120        END
```

SUBROUTINE SUBPROGRAMS

A subroutine subprogram differs from a function subprogram in that a subroutine subprogram is not designed to return one specific value to the calling program. A subroutine generally consists of a set or group of commands which are repeated frequently in the program. By storing the commands in a subroutine, the command set need only be listed once and may be called any number of times from anywhere in the program or subprogram.

Defining A Subroutine Subprogram

A subroutine subprogram always must have a SUBROUTINE declaration as its first statement, an END as the last statement, and at least one RETURN statement. The general form of the SUBROUTINE declaration is as follows:

```
SUBROUTINE      subroutine name      [dummy argument list]
    :
    :
RETURN
    :
    :
END
```

Using arguments in a subroutine subprogram is optional, there need not be any arguments. Including arguments is merely a convenient way of referring to variable values in the calling program or subprogram. All of the variable values in the subprogram and main program are local variable values; that is, they apply only to the subprogram (or main program) in which they were defined and therefore do not exist in any other subprogram unless they are transferred as parameters in the argument list or declared in COMMON using a COMMON declaration. If dummy arguments are used they must be non-subscripted variables or array names. As with dummy arguments in Function Subprograms, no mode is assumed unless it is specifically declared using a type statement in the subroutine, in which case the mode of the actual argument will be compared with the mode of the dummy argument and an error message returned if they are not the same.

Example

```
SUBROUTINE EXPRESSION [X,Y,Z]
Z = (X**2+Y**2)
RETURN
END
```

In this case the value Z was transferred back to the main program as part of the parameter (argument) list. It might instead have been shared with the main program using a COMMON statement.

Calling A Subroutine Subprogram - The CALL Statement

A subroutine subprogram must be called explicitly with the command CALL. It may not be referred to as a function merely by the appearance of its name. The CALL statement has the following general form:

```
CALL      subroutine name      [actual argument list]
```

When the CALL statement is executed, the dummy arguments are assigned the values of the actual arguments and control is transferred to the statement following the SUBROUTINE declaration. The subroutine statements then will be executed in the order indicated until a RETURN statement is encountered which will transfer control out of the subroutine to the statement following the CALL statement.

A subroutine subprogram may be called from the main program or from any subprogram. Subroutines are recursive.

For example, the CALL statement for the previously defined subroutine might appear as follows:

```
CALL EXPRESSION [T-1.5,7.9, ANSWER]
```

The entire program might appear as follows.

```
10          C: MAIN PROGRAM
20          READ (0,50) T
25          CALL EXPRESSION [T-1.5,7.9, ANSWER]
30          WRITE (1,50) ANSWER
35    50    FORMAT (F7.2)
40          END
44          C: SUBROUTINE SUBPROGRAM
45          SUBROUTINE EXPRESSION [X,Y,Z]
55          Z = (X**2+Y**2)
60          RETURN
65          END
```

NOTE: A subroutine subprogram does not return a specific value. Therefore, the subroutine name does not have mode. Do not confuse subroutine and function subprograms.

Example

```
 5          C: MAIN PROGRAM
10          REAL LOW
15          COMMON LOW, HI
20          ACCEPT A,B
25          CALL ABSOLUTE [A,B]
30          SQ = SQRT [HI - LOW]
35          END
40          C: SUBROUTINE SUBPROGRAM
45          SUBROUTINE ABSOLUTE[X,Y]
50          COMMON H1, H2
55          IF (X.GT.Y) GO TO 10
60          H1 = X
65          H2 = Y
70          RETURN
```

```
75    10    H1 = Y
80          H2 = X
85          RETURN
90          END
```

ARGUMENTS

Two types of arguments are found in subprograms, the dummy arguments specified in the definition of the subprogram, and the actual arguments specified when the subprogram is called. Subprogram arguments are a way of passing information (variable values, array values etc.) between the calling program and the subprogram.

Dummy arguments are merely place holders. No space is reserved for them in memory, they have no mode, and, in the case of arrays, they are never dimensioned. Only non-subscripted variables and array names may be used as dummy arguments. When a subprogram is called, the dummy arguments will assume the value, mode, and dimension of the actual arguments listed in the calling statement according to their order in the list. Because the replacement process is position oriented, the number of arguments specified must be the same in both the actual and the dummy argument list. However, the name of the variables and/or arrays used to store the values may be changed.

Although the dummy arguments are not normally considered to have a mode, mode may be assigned to them by including a Type declaration statement within the subprogram. When the dummy arguments are assigned mode, the mode of the actual argument is checked and must agree with the declared mode of the dummy argument or an error message is returned halting program execution.

The actual arguments which are given when the subprogram is called may be variables, array elements, array names (in which case the entire array is shared), constants, or arithmetic expressions. A constant or arithmetic expression is used only to initialize the dummy variables. In this case information may be passed only to the subprogram, any values assigned in the subprogram to the corresponding dummy variable will not be transferred back to the calling program. If the actual argument is specified as a variable, array element, or array name, information may be passed both to and from the subprogram. In this case if the value of the corresponding dummy variable is changed in the subprogram, the value stored in the actual argument will be changed.

RETURN

Every Function and Subroutine subprogram must contain at least one RETURN statement. The RETURN statement is a control command which tells the system to leave the subprogram and return to the calling program.

G. PROGRAM SEGMENTATION

Total program size may be increased from 200 statements to about 600 statements using program segmentation. Program segmentation requires that the total program be broken into a main program and segments which may be called from the main program or from another segment. As many as eight segments may be called provided that the entire program (all segments combined) does not exceed 600 statements and a single segment does not exceed 200 statements. While the program is running, only one segment is in memory at one time. The segments not being used are kept on a high speed storage drum. When a segment is called it is brought into the computer memory and overlays the segment from which it was called. The old segment is stored on the drum and may be recalled if it is needed.

DEFINING A PROGRAM SEGMENT

A program segment is defined by a SEGMENT declaration statement as follows:

```
SEGMENT     segment name
      .
      .
      .
END
```

A segment declaration statement may appear anywhere except in the first or main section of the program. Each segment is considered to be a separate program and may contain one or more subprograms.[1] The main program and each segment must be terminated by an END statement. NOTE: If the end of a segment and the end of a subprogram coincide, only one END statement is required.

A COMMON statement may be used in the segment to transfer variable values to and from the main program and the segments.

CALLING A PROGRAM SEGMENT

A segment may be called in two ways as follows:

```
CALL segment name
        or
GO TO segment name
```

When a CALL command is used to call a segment, control is returned to the statement following the CALL statement after the segment has been executed.

---

[1] NOTE: Subprograms may not be shared by the segments. To use the same subroutine in two segments, the subprogram must be defined in both segments. The COPY command may be used to copy the range of statements desired.

When a GO TO command is used to call a segment, control is not returned to the main program or segment from which the segment was activated.

When a GO TO transfer is used, all variable values from the calling segment not declared in COMMON will be released when the transfer is made to the new segment to allow more storage of variables local to that segment.

FINISH

The execution of a segment may be terminated before reaching the END statement by using a FINISH statement in the segment. Any number of FINISH statements may be placed in a segment.
NOTE: A FINISH statement in the main program will halt execution of the entire program.

```
>LIST

 10.          COMMON A,B,C,AR,SA
 20.          ACCEPT "TYPE SIDES IN DESCENDING ORDER",A,B,C
 30.          AR=AREA[A,B]
 40.          DISPLAY "AREA OF LARGEST SIDE= ",AR
 50.          GO TO VOLUME
 60.          END


 70.          FUNCTION AREA[S1,S2]
 80.          AREA = S1*S2
 90.          RETURN
100.          END


110.          SEGMENT VOLUME
120.          COMMON A,B,C,AR,SA
130.          IF(AR.EQ.0) FINISH
140.          VOL = AR*C
150.          DISPLAY "VOLUME = ",VOL
160.          CALL SUR
170.          DISPLAY "SURFACE AREA = ",SA
180.          DISPLAY "END"
190.          END


200.          SEGMENT SUR
210.          COMMON A,B,C,AR,SA
220.          SA=2*(A*B+A*C+C*B)
230.          END


>RUN
TYPE SIDES IN DESCENDING ORDER4,3,2
AREA OF LARGEST SIDE=     12.
VOLUME =     24.
SURFACE AREA =     52.
END

STOPPED AT: 180.

>
```

H. COMMANDS FROM FILES

A COMMANDS file is a file on which the user may store CCS commands and data which would normally be supplied from the terminal. For example, a COMMANDS file might appear as follows:

```
QUIT↵
COPY /DATA1/ TO TEL↵
COPY /DATA2/ TO TEL↵
CONTINUE↵
CONTINUE↵
1.1, 3.5, 9.4, 3HEND↵
7.996, 34579, GRB↵
```

This file would, when called, return control to the EXECUTIVE, copy the files /DATA1/ and /DATA2/ to the terminal, return to FORTRAN IV, and continue execution of the program.

The FORTRAN IV statement COMMANDS /file name/, which is unique to TYMSHARE, allows the user to open a COMMANDS FROM file during execution of his program. When the COMMANDS file is opened the user will be in the direct (CCS) command mode. The first line of the COMMANDS FROM file must be a direct command or a CCS command. The command CONTINUE will cause the system to resume execution of the program immediately following the COMMANDS statement.

Whenever a COMMANDS file is used to start or continue execution of a program (using the commands RUN or CONTINUE), the input statements ACCEPT and READ(0,...) will take their input automatically from the COMMANDS file rather than from the terminal. A COMMANDS FROM file automatically assumes a file number of zero.

The FORTRAN statement CLOSE(0) closes a COMMANDS file. After the COMMANDS file has been closed, the input again will be taken from the terminal. NOTE: If the user wishes to use terminal input while a COMMANDS FROM file is open, he must open one of the disk files for terminal use as follows:

```
OPEN [3, "TELETYPE", INPUT]
```

COMMANDS FROM files may be created by using the EDITOR commands READ, WRITE, and APPEND, or with the EXECUTIVE command COPY TEL TO /file name/.

CAUTION: Do not confuse the FORTRAN Language statement COMMANDS /file name/ with the CCS command COMMANDS /file name/.

SECTION 4

CCS FORTRAN IV COMMANDS

CCS commands are not part of the FORTRAN IV language, but are features unique to the Tymshare System. The CCS commands make FORTRAN IV programs easy and efficient to create, manipulate, and operate on the Tymshare System.

The user is in the command mode when CCS types a ">". He then may use any of the CCS commands described in this section.

LINE NUMBERS

Each statement of a CCS FORTRAN IV program has a line number. The line number is ignored by the compiler and is independent of any FORTRAN statement label. Line numbers range from .001 to 999.999, inclusive. All FORTRAN statements are compiled and listed according to the order of their line numbers. A segment of a program can be referred to by its range of line numbers. The form either can be

line number

or

line number : line number

A. ENTERING A PROGRAM

a. Entering Statements From The Terminal

ENTER

FORTRAN statements may be entered from the terminal in one of three ways. In all forms of the command, the word "ENTER" is optional.

(1) ENTER With A Line Number

The command:

>ENTER line number FORTRAN statement

allows the user to enter a single FORTRAN statement into his program. The statement must terminate with a Carriage Return. A Line Feed may be used to continue a line. If the line number specified in the command is already the line number of a statement in the program, the statement typed in replaces the original statement. If the line number specified in the command is not already given to a statement in the program, the statement typed in is inserted into the program according to the line number order.

Example

For the program:

```
10.  DIMENSION X(10), Y(10)
20.  INTEGER Z
30.  Z(I) = X(I) + Y(I)
40.  END
```

The commands:

```
>25  DO 10 I=1,10
>ENTER 35 10 CONTINUE
>20  INTEGER Z(10)
```

would produce the program:

```
10.      DIMENSION X(10), Y(10)
20.      INTEGER Z(10)
25.      DO  10  I=1,10
30.      Z(I) = X(I) + Y(I)
35.  10  CONTINUE
40.      END
```

(2) ENTER With A Line Number Range

A range of line numbers can be entered using ENTER as follows:

```
>ENTER  line number:line number
FORTRAN statements
D^C
```

This form of the ENTER command allows the user to enter one or more FORTRAN statements into his program. This form of entering statements terminates with a Control D (D^C). When statement input is terminated, any statements already in the program with the same line numbers as the statements just entered are replaced by the new statements. The statements typed in will be assigned line numbers evenly within the specified range. When this form of the ENTER command is used, CCS prompts an @ sign at the beginning of every line. FORTRAN statements typed in may be separated either by a Carriage Return or semicolon. When a semicolon is used, more than one statement can be entered on a line. A Line Feed is used to continue a line.

Example

For the program:

```
10.  DIMENSION X(10), Y(10)
20.  INTEGER Z
30.  Z(I) = X(I) + Y(I)
40.  END
```

The commands:

```
>20:30
@INTEGER Z(10)
@DO 10 I=1,10
@Z(I) = X(I) + Y(I); 10 CONTINUE D^C
```

produce the program:

```
10.       DIMENSION X(10), Y(10)
20.       INTEGER Z(10)
23.       DO 10 I=1,10
26.       Z(I) = X(I) + Y(I)
29.  10   CONTINUE
40.       END
```

(3) ENTER With Prompted Line Numbers

To enter a range of FORTRAN statements, the user may specify a starting line number, an increment, and a terminating line number in the ENTER command. The form of the command is:

ENTER line number (increment) line number

The following example illustrates this command:

(Remember, the word ENTER is optional.)

```
>10(1)100 ↵
>10 A=5 ↵
>11 B=2*A ↵
>12 X=SQRT[A+B] ↵
>13 Y+SQRT[A-B] ↵
>14 D^C ↵
>
```

The computer responds to the command by first deleting lines 10 to 100. Then as statements are entered from the keyboard, they are numbered beginning with line 10 and incremented by 1 until line 100 is reached. After each statement is entered, the computer prints on the terminal the line number to be assigned to the next statement entered. If the user wishes to terminate statement entry before line 100 is reached, as shown above after line 13, he merely types D^C.

An alternate form of the ENTER command using line number prompting, includes only a starting line number and an increment, as follows:

```
>1(1)
```

In this form, no final line number is specified. This form of the ENTER command is to be used only for initial program entry. When the command is used in this form, statements will be accepted until the user types a D^C. In the example above, line numbering will begin at 10 and increment by 1 until statement entry is terminated by a D^C.

In any of the three forms of the ENTER command, a syntactically incorrect FORTRAN statement will cause a bell to ring; the statement in error will be typed out immediately with an upward arrow pointing to the first character which was not acceptable. The user may then correct the statement.

Example

The command

```
>10 A=B+*D
```

causes the system to type out

```
A=B+*D
    ↑
```

and if the user then types

```
A=B+C*D
```

this statement will be accepted as line 10 in the program.

In all forms of the ENTER command, the word ENTER is optional. Thus the user can create a CCS FORTRAN program merely by typing in each source statement preceded by its intended line number. Lines can be changed, or new lines inserted simply by using line numbers which are identical to or fall between line numbers previously used in the program.

b. Entering Statements From A Symbolic File

The command

```
>COPY /file name/ TO line number:line number
```

causes the FORTRAN statements from the symbolic file specified by the file name to be entered into the program just as with the ENTER command. The file must not have line numbers. Syntactically faulty statements are typed out with the upward arrow error indicator and are then discarded.

c. Entering Statements From Paper Tape

If the user's program is on paper tape, either of two commands may be used to input from paper tape.

If each statement of the program on paper tape has a line number, use the command

```
>LOAD TELETYPE
```

The system will type

```
EACH LINE MUST HAVE A LINE NUMBER.
BEGIN INPUT.
```

Then turn on the paper tape reader. After the tape is read, type a Control D (D^C).

On the other hand, if the statements do not have line numbers punched on the paper tape, use the command

```
>COPY TELETYPE TO line number:line number
```

In this case, the system will type

```
NO LINE MAY HAVE A LINE NUMBER.
BEGIN INPUT.
```

Now turn on the paper tape reader. After the tape is read, type a Control D (D^C). Statements are assigned line numbers within the range specified.

These two commands should be used only with paper tape input because faulty statements are not printed out until Control D is pressed, which prevents immediate keyboard correction.

B. RENUMBERING A PROGRAM

RENUMBER

The user may change the line numbers in his program using any of the forms of the RENUMBER command presented below.

>RENUMBER↵ reassigns line numbers to all statements in the program uniformly in the range 10 to 100.

In the following example, the program was entered using line prompting starting from 1 at increments of 2. The RENUMBER command reassigns line numbers 10 to 100 to the program.

```
>1(2)↵
 1.  C: PROGRAM TO COMPUTE THE AREA OF A TRIANGLE
 3.  C:
 5.  ACCEPT"ENTER VALUES OF A,B,C",A,B,C
 7.  S=(A+B+C)/2.
 9.  AREA=SQRT[S*(S-A)*(S-B)*(S-C)]
11.  WRITE(1,200) A,B,C, AREA
13.  200 FORMAT(4F16.8)
15.  STOP
17.  END

>RENUMBER↵

>LIST↵

10.    C: PROGRAM TO COMPUTE THE AREA OF A TRIANGLE
20.    C:
30.       ACCEPT"ENTER VALUES OF A,B,C",A,B,C
40.       S=(A+B+C)/2.
50.       AREA=SQRT[S*(S-A)*(S-B)*(S-C)]
60.       WRITE(1,200) A,B,C, AREA
70.   200 FORMAT(4F16.8)
80.       STOP
90.       END
>
```

>RENUMBER line range↵
reassigns line numbers uniformly to statements in the line range indicated.

Example

```
>RENUMBER 1.5:47
>RENUMBER Old Line Range AS New Line Range↵
```

reassigns line numbers to statements in the old line range uniformly within the new line range.

Example

```
>RENUMBER 1:20.3 AS 10:50
```

The new line number range must not include any line number belonging to a statement which was not included in the old line number range. This rule is to prevent destroying the rest of the program.

This means that the RENUMBER command cannot be used to erase program lines nor to change the order of statements. For example, in the following:

```
>1    ---
>1.5  ---
>2.3  ---
>7    ---

>RENUMBER 1:2.3 AS 1:6
```

is correct, but

```
>RENUMBER 1:2.3 AS 1:10
```

is not allowed since this would erase statement 7. When the same line number is assigned to two statements by a RENUMBER command, the error message LINE RANGE MISMATCH will be printed.

C. LISTING A PROGRAM

All or part of a program can be listed in formatted form for easy reading, or in unformatted form for quicker printing.

LIST

The command

```
>LIST↵
```

types the entire program in a formatted form with line numbers, statement labels, and statements aligned vertically.

Example

```
>LIST↵
  1.       C: PROGRAM TO COMPUTE THE AREA OF A TRIANGLE
  3.       C:
  5.          ACCEPT"ENTER VALUES OF A,B,C",A,B,C
  7.          S=(A+B+C)/2.
  9.          AREA=SQRT[S*(S-A)*(S-B)*(S-C)]
 11.          WRITE(1,200) A,B,C, AREA
 13.      200 FORMAT(4F16.8)
 15.          STOP
 17.          END
```

Example

```
>LIST 11:15↵
 11.          WRITE(1,200) A,B,C, AREA
 13.      200 FORMAT(4F16.8)
 15.          STOP
>
```

The command

```
>LIST line number↵
```

types out the statement with the specified line number (if one exists) together with its line number.

The command

```
>LIST line number:line number↵
```

lists the part of the program specified by the line number range (if any exists) together with the line numbers.

The user may address any line or range of lines to be listed in a number of ways. He may indicate the position of a line relative to another line.

For example, in the following

```
10.  A=2.7
15.  B=7.1
20.  ACCEPT C
25.  D=A*C+B
30.  DISPLAY D
35.  END
```

the statement ACCEPT C could be specified as line 20, or as line 25-1, or as line 10+2. The latter two methods are examples of relative addressing.

If, for example, the user decides to list lines 15 to 30 in the above program, any of the following forms could be used:

```
>LIST 15:30↵
>LIST 10+1:10+4↵
>LIST 10+1:35-1↵
>LIST 15:35-1↵
```

The command

```
>LIST.↵
```

lists the current line. The period following LIST indicates the current line.

Relative addressing can be used in all commands that allow line addressing.

FAST

An unformatted quick listing can be obtained with the command

```
>FAST line range↵
```

The line range is optional; if it is omitted, the entire program is listed. The following is a listing of the previous example with the command FAST.

```
>FAST↵
 1.  C: PROGRAM TO COMPUTE THE AREA OF A TRIANGLE
 3.  C:
 5.  ACCEPT"ENTER VALUES OF A,B,C",A,B,C
 7.  S=(A+B+C)/2
 9.  AREA=SQRT[S*(S-A)*(S-B)*(S-C)]
11.  WRITE(1,200) A,B,C, AREA
13.  200 FORMAT(4F16.8)
15.  STOP
17.  END
>
```

All of the line addressing methods described under LIST can be used with FAST.

D. DELETING A PROGRAM OR STATEMENTS FROM A PROGRAM

To delete either all or part of a program, the DELETE and CLEAR commands are used.

DELETE

This command will delete parts of a program.

>DELETE line number ↵

deletes the line specified. Thus, >DELETE 35 deletes line 35 from the program.

>DELETE line number:line number ↵

deletes the range of lines specified. Thus, >DELETE 20:40 deletes lines 20 to 40 inclusively.

Relative line addressing can be used with the DELETE command. For example,

>DELETE .	Deletes the current line.
>DELETE .-3:100	Deletes from three lines above the current line up to and including line 100.
>DELETE 10+1:10+4	Deletes from the first line after line 10 up to and including the fourth line after line 10.

CLEAR

This command will erase the entire program. The form of the command is

>CLEAR ↵

The system will reply with

ERASE PROGRAM?

to which the user will answer by typing Y for YES or N for NO. If the answer is Y, the program will be erased and the message OK. typed. If the answer is N, the command will be aborted. In either case, the user receives a command prompt (>).

E. EXECUTING A PROGRAM

To begin execution of a program after it is entered, use the command

>EXECUTE↵

or

>RUN↵

Program execution will start with the first executable statement.

When an input statement is executed, a bell will ring. This means that data is to be typed in.

If an error occurs during execution, the first statement which is in error will be typed with an error diagnostic and execution terminated. The user should correct the error and re-run the program.

F. STORING AND RETRIEVING A PROGRAM

SAVE

After a program has been created, the user may wish to save it on the disk file so he may use it again. The command

```
>SAVE /file name/
```

writes the program with the line numbers on the file specified. The file name is any name that the user chooses to identify his program. Once a program is saved on a file, the user may retrieve it in FORTRAN for execution or modification; or he may look at it in EDITOR (EDITOR is part of the Tymshare system which manipulates text).

If the user wishes to save a copy of his program on paper tape, he may type

```
>SAVE
```

then turn on the paper tape reader and type a Carriage Return. The program is punched on paper tape and listed on the terminal.

LOAD

Once a program has been saved on a file the user may retrieve it any time. To retrieve the program use the command

```
>LOAD /file name/
```

An ALT MODE at this point will abort loading (and hence, preserve the current program). A Carriage Return causes the file to be loaded. If a Carriage Return is pressed, the system will type O.K. and start loading the program specified. When loading is completed, a > is typed.

During loading of a program, faulty statements are printed with an up arrow and discarded. These statements should be corrected before the program is executed.

After a program is loaded, it is ready for execution or modification. If the user has modified his program and wishes to keep the modifications, he should save the program again either under the same file name or under a different file name.

An important use of the LOAD command is to merge files. For example, if the user is creating a program on the terminal, the command >LOAD /A/ will merge by line number the contents of file /A/ with the current program. If the same line number occurs in both the current program and in file /A/, the line in file /A/ will be the one retained. If the line in file /A/ is syntactically incorrect, the line in the current program will be retained.

Files can be merged by the LOAD command as follows. The file /PROG/ contains the following program:

```
10.  ACCEPT X,Y
20.  Z=SQRT[X**2+Y]
30.  THETA=ATAN2[X,Y]
40.  END
```

And the file /CORR/ contains the following:

```
35.  DISPLAY X,Y,Z
20.  Z=SQRT[X**2+Y**2]
```

When both files are loaded by executing two LOAD commands as follows,

```
>LOAD /PROG/
O.K.
>LOAD /CORR/
O.K.
>
```

the following program results:

```
10.  ACCEPT X,Y
20.  Z=SQRT[X**2+Y**2]
30.  THETA=ATAN2 [X,Y]
35.  DISPLAY X,Y,Z
40.  END
```

The two files have been merged by line numbers. Note that the individual files need not be ordered by line numbers as is the case with the file /CORR/.

Since files are merged by the LOAD command, the user can prepare insertions and corrections to a program off line on paper tape and merge the changes with the program to be altered. Thus, the cost of program development is reduced by reducing terminal time.

COPY

This command allows the user to copy a part of a program. The general form of the command is

```
>COPY source TO destination
```

where source and destination can be either files or line ranges or the terminal. The source is always retained.

Examples

>COPY 10:25 TO 75:100↵	Deletes lines 75 to 100 and then inserts the contents of lines 10 to 25 uniformly renumbered in the range 75 to 100. Lines 10 to 75 are undisturbed.

>COPY 1:100 TO /KZZP/↵	Copies lines 1 to 100 to file /KZZP/ without line numbers. Lines 1 to 100 are undisturbed.
>COPY /KZZP/ TO 200:300↵	Deletes lines 200 to 300, then inserts the contents of the file /KZZP/ uniformly numbered in the range 200 to 300. The file /KZZP/ must not have line numbers.
>COPY /FILE A/ TO /FILE B/↵	Copies the contents of /FILE A/ to /FILE B/. The file /FILE A/ may be any type, a symbolic file with line numbers, a binary file without line numbers, or a DUMP file.
>COPY TELETYPE TO 40:50↵	Deletes lines 40 to 50 and inserts the contents of what is typed on the terminal after this command uniformly numbered in the range 40 to 50. Statements typed on the terminal must not have line numbers.

MOVE

The general form of the command is

>MOVE source TO destination

where source is a line range and destination is either a line range or a file name. The MOVE command is similar to the COPY command except that the source is deleted after it is moved.

Examples

>MOVE 10:25 TO 75:100↵	Deletes lines 75 to 100, inserts the contents of lines 10 to 25 uniformly numbered in the range 75 to 100, then deletes lines 10 to 25.
>MOVE 1:100 TO /KZZP/↵	Moves lines 1 to 100 to the file /KZZP/ without line numbers, then deletes lines 1 to 100.

THE EXECUTIVE COMMANDS DUMP AND RUN

Two commands in the EXECUTIVE are described here briefly since they may be used to save and retrieve FORTRAN program.

QUIT

The user may return to the EXECUTIVE from FORTRAN with the command

```
>QUIT↵
-
```

The dash typed by the computer means that the user is returned to the EXECUTIVE.

DUMP

The EXECUTIVE command

```
-DUMP /file name/↵
NEW FILE↵
```

or

```
OLD FILE↵
```

saves the entire contents of the user's program which includes the text of the program, the compiled program, and all data values on a DUMP file. He may reload the file at a later time to continue where he left off.

RUN

The EXECUTIVE command

```
-RUN /file name/ ↵
```

reloads the FORTRAN program that is saved under a DUMP command. The user is returned to FORTRAN and put in the command prompt mode which allows him to do whatever he wishes with his program.

The use of the EXECUTIVE commands DUMP and RUN is the fastest way to save and retrieve FORTRAN programs. It is recommended that the user with large programs use these commands to save time in doing day-to-day debugging. NOTE: If the user wants to look at the contents of the file in EDITOR he must save it with the FORTRAN command SAVE (which saves the symbolic file).

SHRINK

SHRINK is used to avoid excessively large DUMP files. SHRINK eliminates unnecessary information such as segment pages, array storage, and all data values to minimize the amount of disk space used by the program under the EXECUTIVE DUMP command. The sequence of commands used after the user has created a program and wishes to save it as a DUMP file is:

```
>SHRINK↵
>QUIT↵
-DUMP /PROG1/↵
 NEW FILE↵
---------------   intervening work
                  or LOGOUT
-RUN /PROG1/↵
FORTRAN
>
```

LOCK

The LOCK command inhibits all commands but

>RUN or >EXECUTE, >CONTINUE, >QUIT;

and may be used when a program is to be made available as a running program but not to be listed or saved by another user.

The form is

>LOCK↵

The computer types

IS PROGRAM ALREADY SAVED?

to which the user will answer by typing Y for YES or N for NO. If he types Y, the computer will type O.K. and the program is locked. If he types N, the command will be aborted. If other characters are typed, computer will type WHAT? to which he should type Y or N. A command prompt will then be given.

If the user wants to keep a copy of the program he should save it before the LOCK command is used since he will not be able to use the SAVE command after LOCK.

If a locked program is also declared proprietary with the EXECUTIVE command DECLARE, the user is assured complete protection of his program. No one will be able to use it or list it.

COMMANDS

The user may have a set of CCS commands saved on a file. When he uses the command

>COMMANDS /file name/↵

all the commands in the file will be executed.

For example, the file /COM/ has the following instructions:

LIST
RUN
QUIT
FILES
CONTINUE

When the user types

>COMMANDS /COM/↵

his current program will be listed and executed. QUIT returns him to the EXECUTIVE. The files he has in his directory will be listed by the EXECUTIVE FILES command. Then CONTINUE puts him back into FORTRAN IV.

If this is a standard procedure that he wants to go through with each program, all he has to do is to give the commands

```
>LOAD /PROGRAM/↵
 O.K.
>COMMANDS /COM/↵
```

each time he enters the system or each time he runs a program.

G. PROGRAM CONTROL AND DEBUGGING

For debugging and modifying programs, CCS provides the user with the commands REFERENCES and DEFINITIONS. With these commands, the user can refer to the parts of the program he is interested in.

a. To Locate Variables And Label References And Definitions

REFERENCES

The command

>REFERENCES Label or Variable Line Range ↵

will type out all statements together with their line numbers which make reference to the specified variable name or statement label. Declarative occurrences of the identifier are ignored (such as DIMENSION statements, Type Declaration statement, subroutine declaration, etc.). The line range is optional. If a line range is used, all non-declarative statements within the line range will be typed out. If the line range is omitted, all non-declarative statements containing the label or variable specified in the entire program will be typed out.

For example, >REFERENCES 30 will cause the statement labelled 30 to be typed out, and all statements in the entire program with GO TO 30 to be typed out.

>REFERENCES Z 10:50 will cause all statements with Z (except declaration statements) within the line range 10 to 50 to be typed out.

DEFINITIONS

The command

>DEFINITIONS Label or Variable Line Range ↵

will type out all statements together with their line numbers which define the specified variable or statement label. Line range is optional. If it is omitted, all declarative statements containing the label or variable specified are typed out.

For example,

>DEFINITIONS Z will print all declarative references to the variable Z.

>DEFINITIONS Z 10:50 will print declarative references to the variable Z within the line range 10 to 50.

Following are some examples of how the user can make use of the commands REFERENCES and DEFINITIONS for program debugging and modification.

1. Suppose during the execution of a program, the user gets the error message CONFLICTING TYPE DECLARATION with the statement 3.7 REAL X,Y,Z printed. He may type the command

>DEFINITIONS Z ↵

which will type out the following statements:

1.4 SUBROUTINE S[L,M,Z]

3.2 INTEGER T,L,Z

3.7 REAL X,Y,Z

This tells him immediately that he had declared Z to be both INTEGER and REAL. He then could modify either statement 3.2 or 3.7. Suppose he has decided that Z should be real and has kept the statement 3.7 REAL X,Y,Z. In running the program again, he gets the message CONFLICTING TYPE DECLARATION with the statement 3.7 REAL X,Y,Z printed. He may type the command

>REFERENCES S ↵

which will type out the following statements:

9.1 CALL S [5,6,3.9]

12.2 CALL S [5,7,8]

14.5 CALL S [11,2,SQRT[17.4]]

The user immediately finds out that in statement 12.2, he had used the integer 8 in the subroutine CALL when it should have been a real number. He then may change 8 to 8. and re-run the program.

2. Suppose the user has a subroutine in his program called GEM and wishes to add a parameter to all references to the subroutine. He may do this simply by using the command

>REFERENCES GEM ↵

All subroutine calls to GEM will be printed. He then may modify these lines.

3. Suppose the user encounters the error message SUBSCRIPT OUT OF BOUNDS and the statement 30. X=A(35) printed. He may type

>DEFINITIONS A

which types out the statement 10. DIMENSION A(20). He then may modify the statement to reserve more space for the array A.

b. Program Interruption

ALT MODE/ESCAPE

A program may be interrupted at any point during execution by pressing the ALT MODE/ESCAPE key. The computer will type the message INTERRUPTED BEFORE STEP ___, and return the user to the command prompt mode with a > sign. Execution of the program can be resumed from the point of interruption by typing the command

>CONTINUE

c. Breakpoints

BREAK

As many as ten breakpoints can be set easily in the program with the BREAK command as follows:

BREAK 5 8 10:14↵

This command sets a breakpoint at lines 5 and 8 and one breakpoint at every line between 10 and 14 inclusive. NOTE: Type a space between each breakpoint desired. During execution, when a line is encountered where a breakpoint has been set, the following occurs:

1. Execution is interrupted just before executing the statement at the line where the breakpoint was set.
2. The number of the line where the breakpoint was set is printed on the terminal immediately followed by > on the same line, e.g., 10.7>.
3. The user is now in the statement prompt mode (as indicated by a line number and a >).

Execution can be resumed from the breakpoint by typing CONTINUE↵ . To list all active breakpoints, type BREAK↵ . All active breakpoints can be cleared with the >RESET↵ command.

d. Immediate Execution

Once program execution has been interrupted by the ALT MODE key or by encountering a breakpoint, statements can be entered for immediate execution.

To enter a statement, the user merely types @ followed by the statement to be executed, followed by a Carriage Return. When the Carriage Return is depressed, the statement just typed will be compiled immediately, and, if the statement was syntactically correct, executed. In any case, the statement then will be discarded. Any FORTRAN IV statement except a declarative statement may be executed in the immediate mode. CCS commands such as LIST and DELETE may be executed also. Program execution now may be resumed from the point of interruption by typing CONTINUE↵ .

e. Step Execution

After program execution has been interrupted, the user may single step through his program, executing one statement at a time. This is accomplished by typing NEXT↵ for each statement to be executed.

The following program demonstrates some of the debugging features.

```
 LIST↵
100.              Y=.5
110.              Z=.2
120.              LAST=5
130.          10  DO 15 COUNT=1,LAST
140.              Y=Y*SIN[Y]+Y
150.              Z=Z*SIN[Z]+Z
```

```
160.                X=X*SIN[X]+X
170.            15  CONTINUE
180.                A=SQRT[Z**2+Y**2+X**2]
190.                IF(A.GT.1) GO TO 30
200.            20  LAST=LAST*2
210.                GO TO 10
220.            30  DISPLAY LAST,A
230.                END
>BREAK 180 200↵
>EXECUTE ↵
VARIABLE HAS NOT BEEN ASSIGNED A VALUE
160.  X=X*SIN[X]+X
160.  >@X=5↵
160.  >CONTINUE↵
180.  >@DISPLAY X,Y,Z↵
      0.557564177 0.9253519603 0.7916238
180.  >CONTINUE↵
          5      1.339336499
>
```

In this program the user has set two breakpoints, one at Statement 200 and another at Statement 180. He then began execution and encountered an error. The direct statement @X=5 placed the value 5 in the variable X. Execution was resumed at Statement 160 by the command CONTINUE. A breakpoint was encountered at Statement 180 and control returned to the user. The user now asked for X,Y,Z to be displayed by the direct statement @DISPLAY X,Y,Z. Execution was resumed again by the command CONTINUE. The program then typed the answers 5 and 1.339336499 and stopped.

f. Easy Program Checking

Two commands allow the user to verify quickly the executability of his program without actually running it.

CHECK

This command tests for the executability of a program by searching for structural errors such as no END statement in a subroutine or in the main program, labeling or declaration errors, calls to undefined routines, etc. Any error discovered will be printed on the terminal.

Example

```
>CHECK↵
100 A=3
DUPLICATE LABEL
```

If there are no structural errors, the following will be printed:

```
>CHECK↵
 O.K.
>
```

The user now may execute his program by typing RUN.

INITIALIZE

This command, in addition to performing all the functions of the CHECK command, allocates data storage and then begins execution of the program, "breaking" at the first executable statement. The statement with a ">" will be indicated. At this point, the user may enter FORTRAN statements for immediate execution or resume program execution by typing CONTINUE .

Example

```
>INITIALIZE↵
10.>
```

H. EDITING

One of the most powerful features of CCS FORTRAN IV is the ability to edit while in FORTRAN. Editing features allow editing of both program statements and CCS commands. All of the editing characters available in the text editing language EDITOR can be used in FORTRAN.

The following is a summary of the editing control characters:

EDITING CONTROL CHARACTERS		
Control Character	Symbol Printed	Action
		For Deleting
A^c or ←	←	Deletes the previous character typed. Repeated use deletes several characters.
Q^c	↑	Deletes the entire line being typed.
W^c	\	Deletes the preceding word in the line.
P^c and a character	%	Deletes up to but not including the character typed after it.
X^c and a character	%	Deletes up to and including the character typed after it.
K^c		Deletes the next character in the old line, prints the character it deletes.
S^c	%	Deletes the next character in the old line.
M^c or ↵		M^c is the same as a Carriage Return; deletes the rest of the old lines and ends the edit.
		For Copying
D^c		Copies and prints the rest of the old line and ends the edit.
F^c		Copies without printing the rest of the old line and ends the edit.
H^c		Copies and prints the rest of the old line; edit continues at the end of the new line.
Y^c		Copies without printing the rest of the old line; edit continues with the new line acting as old line.
R^c		Prints the rest of the old line plus the new line; edit continues.
T^c		Same as R^c except that it aligns old and new lines.
C^c		Copies the next character in the old line.
O^c and a character		Copies up to but not including the character typed after it.
Z^c and a character		Copies up to and including the character typed after it.

EDITING CONTROL CHARACTERS (Continued)		
Control Character	Symbol Printed	Action
U^C		Copies characters from the old line up to the next tab stop in the new line.
E^C	< >	For Inserting Inserts text into the old line; first E^C prints <, second E^C prints >.
I^C		Others Types spaces up to the next tab stop.
J^C or ↵		J^C is the same as Line Feed; new line continues.
N^C	←	Backs up in old and in new line.

a. Commands Or Statement Input Editing

1. During normal input of CCS commands or FORTRAN statements, A^C, W^C, and Q^C are available.

   A^C Control A or back arrow (←) deletes the previous character typed.

   W^C Deletes the preceding word in the line.

   Q^C Deletes the entire line being typed.

Example

```
20  IF  (X.GRA^C←T.Y)  THEN W^C\GO TO 100
>LIST
20  IF (X.GT.Y) GO TO 100
```

2. If the user types a Control Y (Y^C) at the end of a command or a statement, what he just typed in becomes the old line for editing purposes. All editing characters now can be used.

Example 1

```
>LIFT 1:10 Y^C↵
```

The LIST command is executed.

```
Z^CILIST 1:10↵
```

Example 2

```
>15     A=SQRT[X+Y]Y^C↵
>Z^C+15     A=SQRT[X+Z]↵    He types a Z^C+ which copies the old line up to +, then types Z and ].
```

b. Editing When Syntax Errors Occur

If a syntax error occurs during the ENTER command, the erroneous line with an up arrow will be printed. The line in error is the old line for editing purposes.

Example

```
>10 A=3-*SQRT[3.14159]↵
WRONG NUMBER OF OPERANDS::OPERATORS
A=3-*SQRT[3.14159]
Z^C-A=3-E^C<AE^C>D^C*SQRT[3.14159]
```

The user typed a Z^C followed by '-' which copied the line up to '-'. He then typed an E^C which printed a <; he inserted the variable A, then typed another E^C to end the insertion. To copy the rest of the line, he typed a D^C.

```
>LIST 10
 10.          A=3-A*SQRT[3.14159]
>
```

The corrected line is listed.

If a syntax error occurs during LOAD or COPY /file/ TO line range, the erroneous line is printed, but input continues to be accepted from the file. Thus, bad lines are lost and cannot be edited from the terminal. They must be retyped.

c. Addressing The Lines To Be Edited

To address a line to be edited, type

```
>EDIT line number↵
```

To address a range of lines to be edited, type

```
>EDIT line number:line number↵
```

The EDIT command types out the addressed line (the old line) and allows editing to be done on that line. EDIT also allows editing of a range of lines when a line range is given. Each line is typed out sequentially and made available for editing. When the last line in the line range has been edited, control is returned to the user.

Example

In the following program,

```
10.  ACCEPT A,B,C
20.  S=A-B/2
30.  AREA=SQRT[S*(S+A)*S*(S+B)*S*(S-C)]
40.  WRITE(1,200)  A,B,C,AREA
50.  200 FORMAT(4F10.2)
60.  STOP
70.  END
```

the user edits lines 20 through 50 as follows:

```
>EDIT 20:50 ↵
S=A-B/2 ↵                                   Computer prints line 20 and a Carriage Return.
S=(A+B+C)/2↵                                User types in new line 20 and a Carriage Return
AREA=SQRT[S*(S+A)*S*(S+B)*S*(S-C)] ↵        Computer prints line 30 and a ↵ .
AREA=SQRT[S*(S-A)*(S-B)*P^C(%%%%D^C(S-C)]↵  User types new line up to the * after (S-B), then
                                            a P^C followed by a left parenthesis, which deletes
                                            the old line up to but not including the (.  He
                                            then types in D^C to copy the rest of the line.
WRITE(1,200) A,B,C,AREA↵                    Computer prints the next line.
D^C ↵                                       User decides that this statement is correct and
                                            types a D^C.
200 FORMAT(4F10.2)                          Computer prints the next statement.
Z^C1 200 FORMAT(4F16.8)↵                    User types Z^C1 which copies the old statement up to
                                            the character 1; he then types in the rest of the
                                            line and a ↵ .
```

The new program is listed as below:

```
>LIST
 10.            ACCEPT A,B,C
 20.            S=(A+B+C)/2
 30.            AREA=SQRT[S*(S-A)*(S-B)*(S-C)]
 40.            WRITE(1,200)  A,B,C,AREA
 50.        200 FORMAT(4F16.8)
 60.            STOP
 70.            END
>
```

The command

```
>MODIFY line number↵
```

or

```
>MODIFY line number:line number↵
```

is the same as the EDIT command except that the lines to be edited will not be typed out.

d. Editing Input Data

Data typed into a running program may be edited with the following editing characters:

- A^C Deletes the preceding character and allows it to be retyped.
- W^C Deletes the current field and allows it to be retyped.
- Q^C Restarts the entire input statement from the beginning.

Example

```
>10(10)
10.  DIMENSION A(5)
20.  ACCEPT A
30.  DISPLAY A
40.  END


>RUN
2354A^C←,34.6A^C←78,23.4,56.7W^C↖46.8,10.1
   235.      34.78     23.4     46.8     10.1

STOPPED AT:  30.

>
```

SECTION 5
USER AIDS

A. ABBREVIATED COMMANDS

To save time, any command may be abbreviated to as few characters as is necessary to identify the command uniquely. Thus the command FAST may be abbreviated simply as

```
>F↵
```

since there are no other commands that start with the letter F. But to list a program, at least

```
>LI↵
```

is required since the commands LOAD and LOCK also start with the letter L.

If insufficient identification is given to a command, the message INSUFFICIENT COMMAND IDENTIFIER will be given.

B. COMMAND MODELS

If the user is not certain how to use a particular command, he may type a question mark while entering the command. A model of the command is then typed out.

For example,

```
>COPY?↵
COPY <LINE RANGE OR FILE NAME> TO <LINE RANGE OR FILE NAME>
>
>DEL?↵
DELETE <LINE RANGE>
>
```

To obtain a listing of command models for all the CCS commands type a question mark as follows:

```
>?↵
INITIALIZE (PROGRAM)
EXECUTE
RUN
CONTINUE (EXECUTION)
CLEAR (PROGRAM)
MAP (STORAGE ALLOCATION)
NEXT (STATEMENT)
QUIT (THIS SUBSYSTEM)
CHECK (PROGRAM STRUCTURE)
@ <IMMEDIATE FORTRAN STATEMENT>
SAVE [ <FILE NAME> ]
LOAD [<FILE NAME>]
[ENTER] <LINE NUMBER> <SOURCE STATEMENT> <C.R.>
[ENTER] <LINE>(<INCREMENT>)[<LINE>] <STATEMENTS> <CONTROL D>
[ENTER] <LINE>:<LINE> <STATEMENTS> <CONTROL D>
COMMANDS (FROM) <FILE NAME>
```

```
EDIT <LINE RANGE>
MODIFY <LINE RANGE>
DELETE <LINE RANGE>
LIST [<LINE RANGE>]
FAST [<LINE RANGE>]
COPY <LINE RANGE OR FILE NAME> TO <LINE RANGE OR FILE NAME>
MOVE <LINE RANGE>  TO <LINE RANGE OR FILE NAME>
RENUMBER [ <OLD LINE RANGE> [ AS <NEW LINE RANGE> ] ]
REFERENCES <IDENTIFIER> [<LINE RANGE>]
DEFINITIONS [ <IDENTIFIER> [<LINE RANGE>]
BREAK <MAX OF 10 LINE RANGES>
RESET (BREAKPOINTS)
LOCK
SHRINK
>
```

CAUTION: A space is required between a command and the rest of the command. For example, LOAD/FILE/ will cause an error diagnostic. The user must type LOAD /FILE/.

C. A REVIEW OF CCS PROMPTS

One of the following may be prompted by CCS.

1. A > is a command prompt. When CCS gives a >, it is ready to accept a CCS command.

2. When a line number is printed followed by a >, it means execution of program is continuable at that line. For example, when CCS prints

```
10.>
```

the user may enter any CCS command including an @ followed by a statement for direct execution, or he may type CONTINUE to resume execution of the program at line 10.

3. When CCS prints an @ sign, it is ready to accept a FORTRAN statement. The statement will be included in the line range indicated. This occurs when the user uses the ENTER command with a line number range.

4. When CCS prints a line number, it is ready to accept a FORTRAN statement. This occurs when the user uses the ENTER command in a line prompt mode.

SECTION 6
SAMPLE PROBLEMS

This section contains programs written in FORTRAN IV and executed on the Tymshare system. The problems are presented in an increasing degree of complexity.

MONTHLY PAYMENT PROGRAM

1. DEFINE THE PROBLEM

A. Input

1. Original debt (P)
2. Annual interest (I)
3. Number of monthly payments to be made (N)

B. Compute

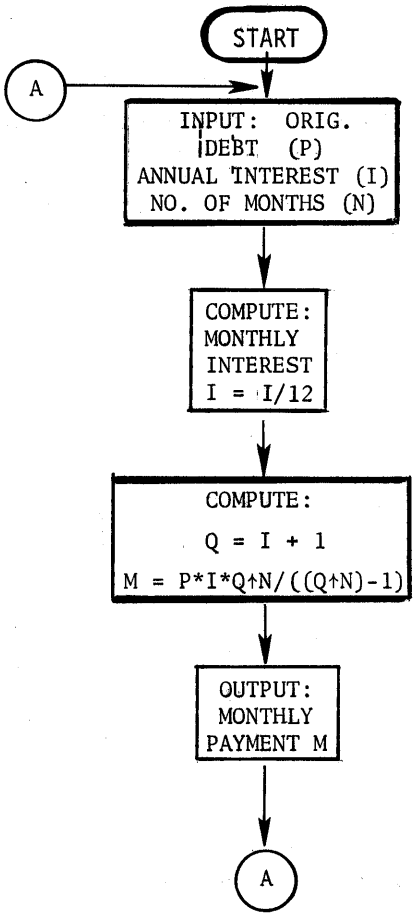
Monthly payment (M)

$$M = \frac{P \cdot I(I+1)^N}{(I+1)^N - 1}$$

C. Output

Monthly payment, M.

2. FLOWCHART


START
A
INPUT: ORIG.
DEBT (P)
ANNUAL INTEREST (I)
NO. OF MONTHS (N)
COMPUTE:
MONTHLY
INTEREST
I = I/12
COMPUTE:
Q = I + 1
M = P*I*Q↑N/((Q↑N)-1)
OUTPUT:
MONTHLY
PAYMENT M
A

3. FORTRAN CODE AND SAMPLE EXECUTION

```
>S(5)
5. REAL I,N,M
10. 7 DISPLAY "***"
15. ACCEPT "PRINCIPAL= $",P,"INTEREST= ",I,"NO.MONTHS= ",N
20. I=I/12
25. Q=I+1
30. M=P*I*Q↑N/((Q↑N)-1)
35. DISPLAY "MONTHLY PAYMENT = $",M
40. GO TO 7
45. END
50.

>RUN
***
PRINCIPAL= $1200
INTEREST= .06
NO.MONTHS= 12
MONTHLY PAYMENT = $    103.2797157
***
PRINCIPAL= $1200
INTEREST= .06
NO.MONTHS= 6
MONTHLY PAYMENT = $    203.514547
***
PRINCIPAL= $1200
INTEREST= .06
NO.MONTHS= 18
MONTHLY PAYMENT = $    69.87807671
***
PRINCIPAL= $
INTERRUPTED BEFORE:
15. ACCEPT "PRINCIPAL= $",P,"INTEREST= ",I,"NO.MONTHS= ",N

15. >SAVE /PR/
NEW FILE


15. >CLEAR
ERASE PROGRAM?Y
OK.

>
```

DOUBLE DECLINING BALANCE DEPRECIATION

1. DEFINE THE PROBLEM

The problem is to compute the double declining balance depreciation on any given asset over any specified number of years using formatted I/O.

A. Input:

1. Cost of the asset (C).
2. Estimated useful lifetime (U).

B. Compute:

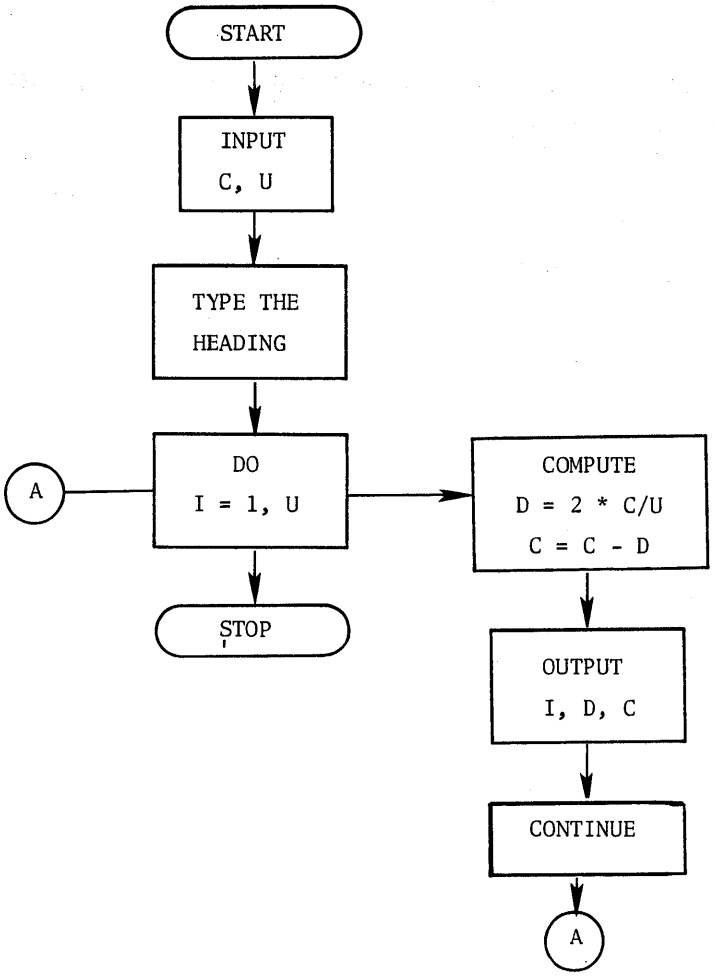
1. Depreciation $D = 2\frac{C}{U}$
2. Book values $C = C - D$

C. Output:

For the entire range of years.

1. Year (I)
2. Amount of depreciation (D)
3. Book value (C)

2. FLOWCHART


START
INPUT
C, U
TYPE THE
HEADING
A
DO
I = 1, U
STOP
COMPUTE
D = 2 * C/U
C = C - D
OUTPUT
I, D, C
CONTINUE
A

3. FORTRAN CODE AND SAMPLE EXECUTION

```
-FORTRAN


>ENTER 10(10)

10. C: DOUBLE DECLINING BALANCE DEPRECIATION PROGRAM
20. WRITE (1,2)
30. 2 FORMAT (//"COST OF ASSET= $",Z)
40. READ (0,3) C
50. 3 FORMAT (F12.2)
60. WRITE (1,4)
70. 4 FORMAT ('ESTIMATED USEFUL LIFETIME= ',Z)
80.  READ (0,3) U
90. WRITE (1,5)
100. 5 FORMAT(/4HYEAR,8X,12HDEPRECIATION,8X,10HBOOK VALUE)
110. DO 100 I=1,U
120. D=2*C/U
130. C=C-D
140. WRITE (1,7) I,D,C
150. 7 FORMAT (I4,8X,"$",F10.2,8X,"$",F8.2)
160. 100 CONTINUE
170. END


>SAVE /DDB/
NEW FILE


>RUN


COST OF ASSET= $3500.00
ESTIMATED USEFUL LIFETIME= 7.

YEAR          DEPRECIATION          BOOK VALUE
   1          $    1000.00         $ 2500.00
   2          $     714.28         $ 1785.71
   3          $     510.20         $ 1275.51
   4          $     364.43         $  911.07
   5          $     260.30         $  650.77
   6          $     185.93         $  464.83
   7          $     132.81         $  332.02

STOPPED AT: 160.

>QUIT

-
```

LEAST-SQUARE LINE

1. DEFINE THE PROBLEM

Fit a least-square line of the form $Y = A + BX$ to the following set of data X,Y where X is the independent and Y the dependent variable.

X	1	3	4	6	8	9	11	14
Y	1	2	4	4	5	7	8	9

Solution:

The work involves computing first the sums shown below.

X	Y	X^2	X Y
1	1	1	1
3	2	9	6
4	4	16	16
6	4	36	24
8	5	64	40
9	7	81	63
11	8	121	88
14	9	196	126
$\Sigma X = 56$	$\Sigma Y = 40$	$\Sigma X^2 = 524$	$\Sigma XY = 364$

Then compute the regression coefficients A and B

$$A = \frac{(\Sigma Y)\ (\Sigma X^2) - (\Sigma X)\ (\Sigma XY)}{N\ \Sigma X^2 - (\Sigma X)^2}$$

$$B = \frac{N\ \Sigma XY - (\Sigma X)\ (\Sigma Y)}{N\ \Sigma X^2 - (\Sigma X)^2}$$

Then $Y = A + BX$

A. Input:

1. Number of data points (N)
2. The X values (X(1) - X(8))
3. The Y values (Y(1) - Y(8))

B. Compute:

Equations given above:

1. Sums needed
2. Coefficients A and B

C. Output:

The equation of the least-square line

2. FORTRAN CODE AND SAMPLE EXECUTION

```
PLEASE LOG IN:
ACCOUNT:   A3
PASSWORD:
USER NAME: MM
PROJ CODE:

READY  4/30  18:30
-FORTRAN


FORTRAN IV A21.00



>10(5)
10. *COMMENT :THIS PROGRAM FITS A LEAST SQUARE
    LINE OF FORM: Y=A+BX TO A SET OF DATA (X,Y)
    WHERE IS THE INDEPENDENT VARIABLE.
15. *
20. DIMENSION X(8),Y(8),XX(8),XY(8)
25. ACCEPT "NUMBER OF DATA POINTS= ",N
30. ACCEPT "THE X VALUES ARE ",X
35. ACCEPT "THE Y VALUES ARE ",Y
40. *COMMENT: LOOP TO CALCULATE XX AND XY
45. *
50. DO 100 I=1,N
55. XX(I)=X(I)↑2
60. XY(I)=X(I)*Y(I)
65. 100 CONTINUE
70. TX=0.
75. TY=0.
80. TXX=0.
85. TXY=0.
90. *COMMENT: LOOP TO CALCULATE TOTALS
95. *
100. DO 200 I=1,N
105. TX=TX+X(I)
110. TY=TY+Y(I)
115. TXX=TXX+XX(I)
120. TXY=TXY+XY(I)
125. 200 CONTINUE
130. *COMMENT: TO CALCULATE COEFFICIENTS A,B
135. A=(TY*TXX-TX*TXY)/(N*TXX-TX*TX)
140. B=(N*TXY-TX*TY)/(N*TXX-TX*TX)
145. WRITE(1,3) A,B
150. 3 FORMAT("THE LEAST SQUARE LINE IS: Y=",F6.3,"+",F6.3,"X")
160. END
```

```
>RUN
NUMBER OF DATA POINTS= 8
THE X VALUES ARE 1,3,4,6,8,9,11,14
THE Y VALUES ARE 1,2,4,4,5,7,8,9
THE LEAST SQUARE LINE IS: Y= 0.545+ 0.636X

STOPPED AT: 150.

>
```

PAYROLL CHECKS (FILE I/O)

1. DEFINE THE PROBLEM

A. Input:

1. Read from file /PRF/ employee number (EMP(I)) and pay rate (RATE(I)).
2. Number of hours worked for each employee.

B. Compute:

1. Gross pay for each employee.

C. Output:

Gross pay file.

File Formats

/PRF/ - Input file

number of employees, Emp_1, $Rate_1$, Emp_2, $Rate_2$, ...

5,99,2.00,77,2.50,88,3.75,55,5.50,66,3.35

/GPF/ - Output file

Emp_1, Pay_1

Emp_2, Pay_2

.
.
.

2. FORTRAN CODE AND SAMPLE EXECUTION

```
-FORTRAN


>LOAD /PAY/
OK.

>LIST

 10.           DIMENSION EMP(200),RATE(200),HRS(200),PAY(200)
 12.         C: OPEN FILE AND READ DATA
 20.           OPEN [3,/PRF/,INPUT,SYMBOLIC]
 30.           READ (3,7) N
 35.          7 FORMAT (I4,Z)
 40.           READ (3,8) (EMP(I),RATE(I),I=1,N)
 45.          8 FORMAT (I3,F7.3,Z)
 50.           CLOSE (3)
 54.         * OPEN FILE,INPUT NUM.HRS., COMPUTE PAY AND OUTPUT TO FILE
 55.           WRITE (1,4)
 56.          4 FORMAT ($ EMP.$,6X,$HRS.$)
 60.           OPEN [4,/GPF/,OUTPUT,SYMBOLIC]
 70.           DO 200 I=1,N
 80.           WRITE (1,9) EMP(I)
 90.          9 FORMAT (I4,8X,Z)
100.           READ (0,6) HRS(I)
110.          6 FORMAT(F6.2)
120.           J=EMP(I)
130.           PAY(J)=HRS(I)*RATE(I)
140.         200 WRITE (4,5) EMP(I),PAY(J)
145.           5 FORMAT (I3,2X,F7.2)
150.           CLOSE (4)
160.           END


>RUN
 EMP.       HRS.
  99         40.
  77         44.4
  88         48.
  55         39.
  66         45.

STOPPED AT: 150.

>@DISPLAY PAY(66)
   150.75


>QUIT
```

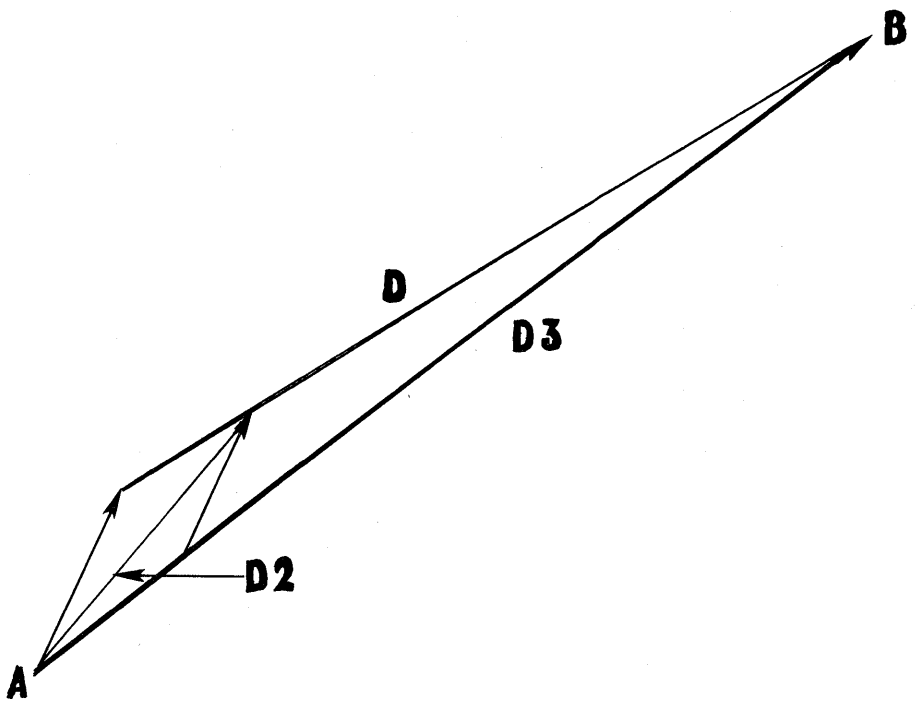
```
-COPY /GPF/ TO TEL

 99     80.00
 77    111.00
 88    180.00
 55    214.50
 66    150.75

-
```

THE DIRECTION OF AN AIRCRAFT

1. DEFINE THE PROBLEM

A plane is flying from point A to point B. The direction is controlled by a radio direction signal from point B. The direction of the plane is corrected for the wind by the autopilot so that the plane is always pointed at B. The problem is to write a program to compute the true distance flown.

Let the heading of the aircraft be constant over some small amount of time. The plane will then follow path D2, compute the new distance D3 to point B. Repeat this until point B is reached. All calculations are done in the complex plane for simplicity.



A. Input:

1. Velocity of aircraft (VM)
2. Heading of aircraft (AA)
3. Velocity of wind (WM)
4. Heading of wind (AW)
5. Distance (DM)

B. Compute:

1. Deviation from flight path (DRM)
2. Total distance traveled (DT)

C. Output:

1. Deviation from flight path (DRM)
2. Total distance traveled (DT)

2. FORTRAN CODE AND SAMPLE EXECUTION

```
10.          COMPLEX D2,D3,V,W,D
12.          DISPLAY"ENTER THE VELOCITY AND HEADING OF THE AIRCRAFT"
14.          ACCEPT VM,AA
20.          DISPLAY"ENTER THE VELOCITY AND DIRECTION OF THE WIND"
22.          ACCEPT WM,AW
28.          DISPLAY"THE DISTANCE OF THE TARGET"
30.          ACCEPT[DM]
32.          RADIAN=360/(2*3.14159)
34.          AW=AW+180
36.          DRM=0
38.          DI=0
40.          AW=AW/RADIAN
42.          AA=AA/RADIAN
44.          AA=AW-AA
46.          WI=WM*SIN[AA]
50.          WR=WM*COS[AA]
52.          W=CMPLX[WR,WI]
54.          V=CMPLX[VM,0]
56.          D1=0
58.          D=CMPLX[DM,0]
62.          DT=0
64.          T=.1
66.          TT=0
70.          K=2
72.          FL=0
74.        3 D2=(V+W)*T
76.          D3=D-D2
84.        5 D1=CABS[D3]
86.          D1=ABS[D1]
88.          DR=REAL[D3]
90.          IF(DI.LT.DRM) GO TO 50
92.       40 DRM=DI
96.       50 TT=TT+T
98.          DI=AIMAG[D3]
100.         DM=CABS[D3]
102.         D2M=CABS[D2]
104.         DT=DT+D2M
106.         VM=CABS[V]
108.         VR=VM*DR/DM
110.         VI=VM*DI/DM
112.         V=CMPLX[VR,VI]
114.         D=D3
116.         A=VM*T
120.         IF(D1.GT.A) GO TO 3
124.      12 T=T/10
126.         FL=FL+1
128.         IF(FL.NE.K) GO TO 3
132.      10 DISPLAY" LARGEST DEVIATION FROM THE FLIGHT PATH=",DRM
136.     585 DISPLAY"THE TOTAL DISTANCE TRAVELED WAS ",DT
138.         END
```

```
>RUN
ENTER THE VELOCITY AND HEADING OF THE AIRCRAFT
600,45
ENTER THE VELOCITY AND DIRECTION OF THE WIND
200,135
THE DISTANCE OF THE TARGET
2000
 LARGEST DEVIATION FROM THE FLIGHT PATH=   256.3855477
THE TOTAL DISTANCE TRAVELED WAS    2116.221673

STOPPED AT: 136.
```

CHECKING ACCOUNT PROGRAM

In this problem, we wish to compute the monthly service charge for a regular checking account. The amount of the service charge is based on the average monthly balance and the number of checks written. The charge may be computed from the following table.

AVERAGE MONTHLY BALANCE

NUMBER OF CHECKS	UNDER $200	$200 to $299	$300 to $399	$400 to $499	$500 to $599	$600 to $699	$700 to $799	$800 to $899	$900 to $999	$1000 to $1099	$1100 to $1199	$1200 to $1299	$1300 to $1399	$1400 to $1499	$1500 to $1599
0	$ .75	$ .47	$ .33	$	$	$	$	$	$	$	$	$	$	$	$
1	.82	.54	.40												
2	.89	.61	.47	.33											
3	.96	.68	.54	.40											
4	1.03	.75	.61	.47											
5	1.10	.82	.68	.54											
6	1.17	.89	.75	.61											
7	1.24	.96	.82	.68	.54										
8	1.31	1.03	.89	.75	.61										
9	1.38	1.10	.96	.82	.68	.54									
10	1.45	1.17	1.03	.89	.75	.61									
11	1.52	1.24	1.10	.96	.82	.68	.54								
12	1.59	1.31	1.17	1.03	.89	.75	.61								
13	1.66	1.38	1.24	1.10	.96	.82	.68	.54							
14	1.73	1.45	1.31	1.17	1.03	.89	.75	.61							
15	1.80	1.52	1.38	1.24	1.10	.96	.82	.68	.54						
16	1.87	1.59	1.45	1.31	1.17	1.03	.89	.75	.61						
17	1.94	1.66	1.52	1.38	1.24	1.10	.96	.82	.68	.54					
18	2.01	1.73	1.59	1.45	1.31	1.17	1.03	.89	.75	.61					
19	2.08	1.80	1.66	1.52	1.38	1.24	1.10	.96	.82	.68	.54				
20	2.15	1.87	1.73	1.59	1.45	1.31	1.17	1.03	.89	.75	.61				
21	2.22	1.94	1.80	1.66	1.52	1.38	1.24	1.10	.96	.82	.68	.54			
22	2.29	2.01	1.87	1.73	1.59	1.45	1.31	1.17	1.03	.89	.75	.61			
23	2.36	2.08	1.94	1.80	1.66	1.52	1.38	1.24	1.10	.96	.82	.68	.54		
24	2.43	2.15	2.01	1.87	1.73	1.59	1.45	1.31	1.17	1.03	.89	.75	.61		
25	2.50	2.22	2.08	1.94	1.80	1.66	1.52	1.38	1.24	1.10	.96	.82	.68	.54	.00

A. Input:

1. Number of checks written (NUMCHKS)
2. Average monthly balance (AVGBAL)
3. Current monthly balance (CURBAL)

B. Compute;

1. This month's service charge (MATRIX (I,J))
2. New balance (CURBAL)

C. Output:

1. This month's service charge
2. New balance.

2. FORTRAN CODE AND SAMPLE EXECUTION

```
LIST


 10.     C:       CHECKING ACCOUNT
 20.     C:;
 30.          DISPLAY 'THIS IS A PROGRAM TO COMPUTE THE MONTHLY SERVI
             CE CHARGE FOR A REGULAR   CHECKING ACCOUNT AT A COMMERCIAL B
             ANK.';
 40.     C:;
 50.          REAL MATRIX, NUMCHKS;
 60.          DIMENSION MATRIX(15,26);
 70.          WRITE(1,50);
 80.     50   FORMAT(//'     CURRENT BALANCE = ',Z);
 90.     75   FORMAT(/'     AVERAGE BALANCE FOR THIS MONTH = ',Z);
100.     80   FORMAT(/'     NUMBER OF CHECKS THIS MONTH = ',Z);
110.          ACCEPT CURBAL;
120.          DO 100, I=1,15;
130.          IF(I.LT. 15) MATRIX(I,26) = 2.50 - (I*.14);
140.          IF(I.EQ. 15) MATRIX(I,26) = 0.00;
150.          IF(I.EQ. 1) MATRIX(I,26) = 2.50;
160.     100  CONTINUE;
170.          DO 200, I=1,15;
180.          DO 200, J=1,25;
190.          MATRIX(I,J) = MATRIX(I,26)-((26-J)*.07);
200.     200  CONTINUE;
210.          WRITE(1,75);
220.          ACCEPT AVGBAL;
230.          WRITE(1,80);
240.          ACCEPT NUMCHKS;
250.          I = AVGBAL/100;
260.          J = NUMCHKS + 1;
270.          IF(MATRIX(I,J) .GE. 0.54) GO TO 400;
280.          MATRIX(I,J) = 0.00;
290.          IF ((I.EQ.2.0.AND.J.EQ.1.0).OR.(I.EQ.3.0.AND.J.EQ.3.0)
                 .OR.(I.EQ.4.0.AND.J.EQ.5.0)) MATRIX(I,J) = 0.47;
300.          IF ((I.EQ.3.0.AND.J.EQ.2.0).OR.(I.EQ.4.0.AND.J.EQ.4.0))
                  MATRIX (I,J) = 0.40;
310.          IF ((I.EQ.3.0.AND.J.EQ.1.0).OR.(I.EQ.4.0.AND.J.EQ.3.0))
                  MATRIX (I,J) = 0.33;
320.     400  DISPLAY 'THIS MONTHS SERVICE CHARGE =',MATRIX(I,J) ;
330.          CURBAL = CURBAL - MATRIX(I,J);
340.          DISPLAY 'THE NEW CURRENT BALANCE =',CURBAL;
350.          END;

>EXECUTE
THIS IS A PROGRAM TO COMPUTE THE MONTHLY SERVICE CHARGE FOR A REGULAR
CHECKING ACCOUNT AT A COMMERCIAL BANK.
```

```
     CURRENT BALANCE =  513.67

     AVERAGE BALANCE FOR THIS MONTH = 336.71

     NUMBER OF CHECKS THIS MONTH = 13
THIS MONTHS SERVICE CHARGE =    1.24
THE NEW CURRENT BALANCE =     512.43
```

SPECTRAL EFFICIENCY OF A BLACK BODY

The distribution of energy emitted from a black body, with respect to wave length is given by Planck's distribution law:

$$E(\lambda)\, d\lambda = \frac{C_1}{\lambda^5} \left(e^{C_2/\lambda T} - 1\right)^{-1} d\lambda \qquad (1)$$

where C_1 and C_2 are the first and second Planck radiation constants.

$$C_1 = 8\pi hc = (4.9918 \pm 0.002) \times 10^{-15} \text{ erg-cm.}$$
$$C_2 = hc/k = 1.43880 \pm 0.0007 \text{ cm-deg.}$$

The total density of radiant energy in unit volume is (1), integrated over all wave lengths.

$$E_{Total} = \frac{8\pi (kT)^4}{h^3c^3} \int_0^\infty \frac{u^3 du}{e^u - 1} = \frac{8\pi^5 (kT)^4}{15h^3c^3} = a\, T^4 \qquad (2)$$

$$\text{where } a = \frac{8\pi^5 k^4}{15h^3c^3}$$

(2) is known as the Stefan-Boltzmann law.

If we consider radiant energy in terms of flux, the total radiation crossing a unit area in unit time in all directions in one hemisphere is usually written:

$$E_{total} = \sigma T^4$$

$$\text{where } \sigma \text{ (Stefan-Boltzmann Constant)} = \frac{2\pi^5 k^4}{15h^3c^3} = 5.672 \times 10^{-5} \frac{\text{erg}}{\text{cm}^2\text{deg}^4}$$

For practical calculations this reduces to:

$$E_{total} = (T/645)^4$$

E = total radiant flux in watts/cm^2
T = Temperature in degrees K.

If we integrate Planck's equation between the wavelength interval in which we are interested and compare this to the total radiation, we can obtain the relative efficiency of a black body emitter, for a selected wavelength interval, as a function of temperature.

$$E_{partial} = b \int_{x_1}^{x_2} x^5 \left(e^{1.4388/Tx} - 1\right)^{-1} dx \qquad \text{where } x = \text{wavelength in cm.}$$

For $E_{partial}$ in watts/cm^2 $\quad b \simeq 2.39 \times 10^{-11}$

Relative Efficiency = E = $E_{partial}/E_{total} \times 100$

which reduces to

$$E_{(T)} = 64.774 \int_{x_1}^{x_2} X^5 \, (e^{1.4388/Tx_{-1}})^{-1} \, dx/T^4$$

The integral can be evaluated by Simpson's 1/3 rule as follows:

$$E = \int_{x_1}^{x_2} f(x)dx = \frac{h}{3} \, [f(x_1) + 4 f (x_1+h) + 2 f(x_1 + 2h)$$

$$+ 4 f (x_1 + 3h) + \text{---} + 2 f (x_2 - 2h)$$

$$+ 4 f (x_2 - h) + f (x_2)]$$

where $h = \frac{x_2 - x_1}{n}$ where n = number of terms in the series; n is even.

We can evaluate the integral easily by separately summing the terms multiplied by 4 and the terms multiplied by 2 and then adding in the first and last terms.

Since the Planck function is used in several places, it is convenient to use the DEFINE statement.

If the user uses an n = 100, he will get results accurate to the five significant digits printed. In fact, n = 10 is quite accurate for temperatures over 5000° K. The reader may want to experiment with this. The following chart shows some sample results of such experimentation.

SPECTRAL EFFICIENCY PROGRAM

Test for sensitivity to N over visible spectrum

$X_1 = 4.0 \times 10^{-5}$ $X_2 = 7.0 \times 10^{-5}$

N =	1000	100	10	2
TEMP.				
500°	1.5850-12	1.5850-12	1.6222-12	4.3237-12
1000°	1.8041-04	1.8041-04	1.8054-04	2.3938-04
5000°	3.1033 01	3.1033 01	3.1034 01	3.1065 01
10000°	3.2117 01	3.2117 01	3.2117 01	3.2040 01
15000°	1.8245 01	1.8245 01	1.8245 01	1.8315 01
20000°	1.0464 01	1.0464 01	1.0464 01	1.0554 01

1. DEFINE THE PROBLEM

A. Input:

1. Upper and lower wave length boundaries (X1, X2).
2. Temperature ranges in degrees Kelvin (T1, T2).
3. The temperature interval (T3).
4. The number of integration intervals (N).

B. Compute:

The spectral efficiency of a black body.

C. Output:

1. Temperature (T)
2. Percent efficiency (E)

2. FORTRAN CODE AND SAMPLE EXECUTION

```
>RUN
WAVELENGTH(CM) RANGES FROM X1= 4.0E-5 TO X2= 7.0E-5
TEMPERATURE(DEGREES K) RANGES FROM!
 T1=500 TO T2=20000 IN STEPS OF T3=500
EVEN NUMBER OF INTEGRATION N=20

         TEMPERATURE          PERCENT EFFICIENCY
                500                  .1587E-11
               1000                  .1804E-03
               1500                  .5453E-01
               2000                  .7629E+00
               2500                  .3274E+01
               3000                  .7941E+01
               3500                  .1402E+02
               4000                  .2043E+02
               4500                  .2626E+02
               5000                  .3103E+02
               5500                  .3455E+02
               6000                  .3688E+02
               6500                  .3815E+02
               7000                  .3857E+02
               7500                  .3832E+02
               8000                  .3758E+02
               8500                  .3649E+02
               9000                  .3515E+02
               9500                  .3367E+02
              10000                  .3211E+02
              10500                  .3052E+02
              11000                  .2893E+02
              11500                  .2738E+02
              12000                  .2587E+02
              12500                  .2442E+02
              13000                  .2304E+02
              13500                  .2174E+02
              14000                  .2050E+02
              14500                  .1934E+02
              15000                  .1824E+02
              15500                  .1721E+02
              16000                  .1625E+02
              16500                  .1535E+02
              17000                  .1450E+02
              17500                  .1371E+02
              18000                  .1297E+02
              18500                  .1228E+02
              19000                  .1163E+02
              19500                  .1103E+02
              20000                  .1046E+02
PAUSE

150. >
```

-

-FORTRAN

```
>LOAD /PR/
OK.

>LIST

 10.          3 ACCEPT "WAVELENGTH(CM) RANGES FROM X1= ",X1,"TO X2= ",X2
 20.            ACCEPT "TEMPERATURE(DEGREES K) RANGES FROM T1=",T1,"TO T2=
                ",T2,"IN STEPS OF T3=",T3
 30.            ACCEPT "EVEN NUMBER OF INTEGRATION N=",N
 40.            IF (MOD[N,2].NE.0) N=N+1
 50.            H=(X2-X1)/N
 60.            WRITE (1,9)
 70.          9 FORMAT (/8X,$TEMPERATURE$,10X,$PERCENT EFFICIENCY$)
 80.            F[A,B]=1.0/(A↑5*(EXP[1.4388/(B*A)]-1))
 90.            DO 100 T=T1,T2,T3
100.            E=(64.774)*((H/3)*(F[X1,T]+4*(SUM[X1+H,X2-H,2*H,T])+2*(SUM
                [X1+2*H,X2-2*H,2*H,T])+F[X2,T]))/T↑4
110.            WRITE (1,10) T,E
120.         10 FORMAT (13X,I5,16X,E11.4)
130.        100 CONTINUE
140.            PAUSE
150.            GO TO 3
160.            END


170.            FUNCTION SUM[W,X,Y,T]
180.            S=0
190.            F[A,B]=1.0/(A↑5*(EXP[1.4388/(B*A)]-1))
200.            DO 200 R=W,X,Y
210.            S=F[R,T]+S
220.        200 CONTINUE
225.            SUM=S
230.            RETURN
240.            END
```

THE BASE OF A SEMICIRCLE

1. DEFINE THE PROBLEM

Given n number of chords inscribed within a semicircle and their respective lengths, find the diameter of the semicircle.

If we construct triangles with the radii of the semicircle and bisect each chord length with a construction line, C, we have n half angles, $\phi_{(n)}$, whose sum is equal to 90°, or $\pi/2$ radians; from the sum of the angles equal to 180°. First, we approximate the angle ϕ:

$$\phi = [(\text{side(I)}/2) \ / \text{sum of sides}]$$

We also know that the angle β is equal to 90°, from this we may approximate the diameter using the law of sines;

$$(\text{diameter}/2)/\sin \ \phi = (\text{side(I)}/2)/\sin \phi$$

Knowing that the sine of 90° is equal to 1.0000 we can see:

$$\text{diameter} = 2[(\text{side(I)}/2)/\sin \phi]$$

Since our first calculation of ϕ was only a rough approximation and thus the diameter was only a rough approximation, we now want to approximate the diameter to a greater degree of accuracy.

We know that when the sum of the half angles is equal to $\pi/2$, or 1.57079, we have a very close approximation of the diameter of the semicircle. We may approximate the half angles by using the tangent and arctangent functions.

We know that;

$$\tan = \frac{\text{side(I)}/2}{C \quad \text{(construction line)}}$$

since,

$$C = (\text{diameter}/2)^2 - (\text{side(I)})^2$$

Therefore:

$$\tan = \frac{(\text{side(I)}/2)}{[(\text{diameter}/2)^2-(\text{side(I)})^2]^{1/2}}$$

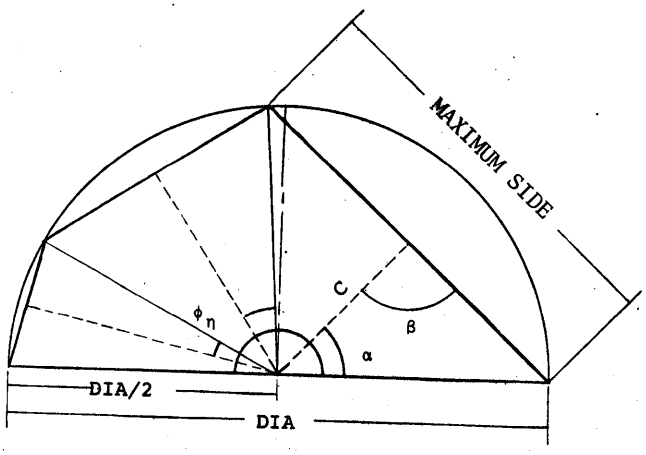
and that:

$$= \tan^{-1} \frac{(\text{side(I)}/2)}{[(\text{diameter}/2)^2-(\text{side(I)}/2)^2]^{1/2}}$$

From the above calculation, if the sum of the angles is greater than $\pi/2$, or 1.57079, the diameter must be made larger by some increment. If the sum of the angles is equal to $\pi/2$, the approximation

is exact to five decimal places, and if the sum of the angles is less than $\pi/2$, the approximation of the diameter is too large and must be reduced by some increment.

Following is a program to solve the problem by incremental approximations of the diameter.



A. Input:

1. Number of sides (N)
2. The sides SIDE(N)

B. Compute:

Do computations explained above.

C. Display:

1. Length of the diameter.

2. FORTRAN CODE AND SAMPLE EXECUTION

```
C:    BASE OF A SEMI-CIRCLE
C:    GIVEN: THE NUMBER OF SIDES AND THEIR LENGTHS
      ACCEPT "    THE NUMBER OF SIDES = ",N
        CALL CALC(N)
        END
        SUBROUTINE CALC(N)
        DIMENSION SIDE(N)
        REAL INCR
        SUMSID = 0
        CNTR = 0
        ALPHA = 0
        MAX = 0
        MIN = 1
        DO 30, I=1,N
      ACCEPT "    SIDE = ",SIDE(I)
        SUMSID = SUMSID+SIDE(I)
        MAX = AMAX1(MAX,SIDE(I))
     30 CONTINUE
        ALPHA = ((MAX/2)/SUMSID)*3.14159
        DIA1 = (MAX/2)/SIN(ALPHA)
        DIA = DIA1*2
        INCR = MAX/10
    100 ANGLE = 0
        DO 40, I=1,N
        Y = SIDE(I)
        X = SQRT((DIA**2)-(SIDE(I)**2))
     40 ANGLE = ANGLE + ATAN(Y/X)
        IANGLE = ANGLE*100000
        IF(157079-IANGLE)50,60,70
     50 DIA = DIA + INCR
        GO TO 100
     70 DIA = DIA - INCR
        INCR = INCR/10
        DIA = DIA + INCR
        CNTR = CNTR + 1
        IF (CNTR-20)100,100,60
     60 DISPLAY '     THE BASE OF THE SEMI-CIRCLE =' ,DIA
        RETURN
        END
```

```
>EXECUTE
     THE NUMBER OF SIDES = 3
     SIDE = 1
     SIDE = 2
     SIDE = 3
     THE BASE OF THE SEMI-CIRCLE =     4.113103501

>EXECUTE
     THE NUMBER OF SIDES = 2
     SIDE = 1
     SIDE = 1
     THE BASE OF THE SEMI-CIRCLE =     1.4142145

>
```

APPENDIX A

EFFICIENT STORAGE ALLOCATION

The computer has a specified amount of memory available for both program and data. By segmenting your program; that is, storing only part of it in memory and the rest on the drum (another data storage device), the total number of statements and amount of data in a program may be increased greatly.

A maximum of four pages of memory is available for both the program (segment) and the data. The amount of storage needed for data must be estimated. The amount of storage required for each program segment may be found using the CCS command MAP.

A page may not be shared by the program statements and the data. Therefore if part of a page is used for data storage, the entire page must be considered filled.

The maximum size of the segment may be estimated as follows: Maximum Segment Size = 4 pages minus the data storage requirements. Data storage needs may be estimated using the following values:

1 Page = Approximately 2000 Words

1 Real Number = 2 Words

1 Integer Number = 2 Words

1 Complex Number = 4 Words

1 Double Precision Number = 3 Words

Thus, if a program contains the real array A(130,20), 5200 words or 3 pages would be required for data storage leaving one page for program segment storage.

The actual segment size may be determined using the CCS command MAP. The MAP command prints a table as shown in the example below.

```
>MAP


TEXT PAGES = 2.8
UNUSED PAGES = 3
SEGMENT ........     0     1     2     3
STATEMENTS .....   150   175   127   130
PAGES USED .....   0.6   0.8   0.5   0.9
>
```

If any of the segments exceeds the maximum size allowed, they must be reduced in size before the program will run.

APPENDIX B

INTERNAL REPRESENTATION OF ASCII CODE

Octal Representation	Character	Octal Representation	Character
00		40	@
01	!	41	A
02	"	42	B
03	#	43	C
04	$	44	D
05	%	45	E
06	&	46	F
07	'	47	G
10	(	50	H
11	)	51	I
12	*	52	J
13	+	53	K
14	,	54	L
15	-	55	M
16	.	56	N
17	/	57	O
20	0	60	P
21	1	61	Q
22	2	62	R
23	3	63	S
24	4	64	T
25	5	65	U
26	6	66	V
27	7	67	W
30	8	70	X
31	9	71	Y
32	:	72	Z
33	;	73	[
34	<	74	\
35	=	75	]
36	>	76	↑
37	?	77	←

SUMMARY - FORTRAN IV LANGUAGE STATEMENTS

ABBREVIATIONS

ln. = line number
var = variable
exp. = expression
vl. = variable list
sn. = statement number
val = value
arg. list = argument list
< > = anything enclosed in < > is optional.

CONSTANTS

Integer e.g. 1, 400
Real e.g. 1.9, 1.7E 07, 5E-55
Complex (3,5.2)
Logical .TRUE., .FALSE.
Hollerith 4HTIME, "XXX", 'RED', RTM

ARITHMETIC OPERATORS (In Order Of Priority)

↑ or **	Exponentiation
-	Unary minus (negative)
*, /	Multiplication, Division
+, -	Addition, Subtraction

RELATIONAL OPERATORS

.EQ.	Equal to
.NE.	Not equal to
.LT.	Less than
.LE.	Less than or equal to
.GT.	Greater than
.GE.	Greater than or equal to

LOGICAL OPERATORS (In Order of Priority)

.NOT.
.AND.
.OR.
.EOR.

ASSIGNMENT STATEMENTS

variable = expression

INPUT/OUTPUT STATEMENTS

ACCEPT vl, "text", $text$, 'text'

DISPLAY vl, "text", $text$, 'text', expression

Formatted I/O (symbolic)

READ or WRITE (file no. , format no.) list

File Numbers
0 = Terminal Input
1 = Terminal Output
2,3,4 = disk file I/O

Open & Close Disk File

OPEN [file no. , /file name/ , INPUT or OUTPUT , SYMBOLIC or BINARY]

CLOSE (file no.)

Binary I/O

READ or WRITE (file no.) list

Commands From Files

COMMANDS /file name/

CONTROL STATEMENTS

IF (logical exp) executable statement

IF (arithmetic exp) sn_1, sn_2, sn_3

DO sn var = init val., final val, interval>

GO TO<sn>or<var>or<segment name>

GO TO (sn_1, sn_2, sn_3...) var

CALL<subroutine name [actual arg. list]> or <segment name>

ASSIGN integer TO var

CONTINUE RETURN

PAUSE FINISH

STOP END

DECLARATION STATEMENTS

C:
* comments

DATA $var_1/val_1/,var_2/val_2/...$
DATA $var_1, var_2,.../val_1, val_2,.../$

DIMENSION array name (lower limit$_1$: upper limit$_1$, lower limit$_2$: upper limit$_2$,...)

INTEGER
REAL
DOUBLE PRECISION
COMPLEX
LOGICAL
v1 or label

function name [dummy arg. list] = 1

FUNCTION function name [dummy arg. list]

SUBROUTINE subroutine name < [dummy arg. list] >

SEGMENT segment name

COMMON v1.

format no. FORMAT (field spec)

SUMMARY - CCS FORTRAN IV COMMANDS

Line number = .001 - 999.999

```
<ENTER>  [ ln  FORTRAN statement
         | ln:ln        (DC terminator)
         | ln (interval)<ln>    (DC terminator)

COPY  [ /file name/ ]     TO  [ /file name/ ]
      [   ln:ln     ]         [   ln:ln     ]
      [ TELETYPE    ]         [ TELETYPE    ]

MOVE  ln:ln  TO  [ /file name/ ]
                 [   ln:ln     ]
                 [ TELETYPE    ]

RENUMBER  [       (starts at 100 in steps of 10)
          | ln:ln
          | ln:ln AS ln:ln

LIST      [           (entire program)
FAST      | ln
          | ln:ln
DELETE    |.          (current line)

REFERENCES    [label      <ln:ln>
DEFINITIONS   [or variable

EDIT      [ln
MODIFY    [ln:ln

BREAK ln  ln:ln       (maximum of 10 lines or line ranges)
RESET

CHECK                 RUN          SAVE /file name/
INITIALIZE            EXECUTE      LOAD /file name/
SHRINK                NEXT         COMMANDS /file name/
LOCK                  CLEAR
MAP                   QUIT
@FORTRAN statement    CONTINUE
```

TYMSHARE MANUALS

Instant Series

CAL
SUPER BASIC
EDITOR

Reference Manuals

EXECUTIVE
CAL
SUPER BASIC
EDITOR
FORTRAN IV
FORTRAN II
LIBRARY
COGO
ECAP
ARPAS/DDT
BRS

TYMSHARE™

TYMSHARE, INC.

SAN FRANCISCO
745 Distel Drive
Los Altos, California 94022
Telephone: 415/961-0545

LOS ANGELES
334 East Kelso Street
Inglewood, California 90301
Telephone: 213/677-9142

SAN DIEGO/ORANGE COUNTY
4630 Campus Drive, Suite 209
Newport Beach, California 92660
Telephone: 714/540-5940

NEW YORK
464 Hudson Terrace
Englewood Cliffs, New Jersey 07632
Telephone: 201/567-9110

DALLAS
2355 Stemmons Bldg., Suite 1010
Dallas, Texas 75207
Telephone: 214/638-5680

SEATTLE
2200 6th Avenue, Suite 810
Seattle, Washington 98121
Telephone: 206/MA 3-8321