296 fog
2/28
AEC RESEARCH AND DEVELOPMENT REPORT

MASTER

TERMTRAN

A PROGRAMMING SYSTEM FOR
ENGINEERING CALCULATIONS
VIA REMOTE TERMINALS

AUTHORS:

G. B. Knight
R. E. Neu

**UNION
CARBIDE**

UNION CARBIDE CORPORATION
NUCLEAR DIVISION

Operating the

OAK RIDGE GASEOUS DIFFUSION PLANT • OAK RIDGE NATIONAL LABORATORY
OAK RIDGE Y-12 PLANT • PADUCAH GASEOUS DIFFUSION PLANT

*For the Atomic Energy Commission
Under U.S. Government Contract W7405 eng 26*

DISTRIBUTION OF THIS DOCUMENT IS UNLIMITED

DISCLAIMER

This report was prepared as an account of work sponsored by an agency of the United States Government. Neither the United States Government nor any agency Thereof, nor any of their employees, makes any warranty, express or implied, or assumes any legal liability or responsibility for the accuracy, completeness, or usefulness of any information, apparatus, product, or process disclosed, or represents that its use would not infringe privately owned rights. Reference herein to any specific commercial product, process, or service by trade name, trademark, manufacturer, or otherwise does not necessarily constitute or imply its endorsement, recommendation, or favoring by the United States Government or any agency thereof. The views and opinions of authors expressed herein do not necessarily state or reflect those of the United States Government or any agency thereof.

DISCLAIMER

Portions of this document may be illegible in electronic image products. Images are produced from the best available original document.

Printed in the United States of America. Available from Clearinghouse for Federal
Scientific and Technical Information, National Bureau of Standards,
U.S. Department of Commerce, Springfield, Virginia 22151
Price: Printed Copy \$3.00; Microfiche \$0.65

LEGAL NOTICE

This report was prepared as an account of Government sponsored work. Neither the United States, nor the Commission, nor any person acting on behalf of the Commission:

- A. Makes any warranty or representation, expressed or implied, with respect to the accuracy, completeness, or usefulness of the information contained in this report, or that the use of any information, apparatus, method, or process disclosed in this report may not infringe privately owned rights; or
- B. Assumes any liabilities with respect to the use of, or for damages resulting from the use of any information, apparatus, method, or process disclosed in this report.

As used in the above, "person acting on behalf of the Commission" includes any employee or contractor of the Commission, or employee of such contractor, to the extent that such employee or contractor of the Commission, or employee of such contractor prepares, disseminates, or provides access to, any information pursuant to his employment or contract with the Commission, or his employment with such contractor.

Report Date: February 23, 1968

Report Number: K-1735

Subject Category: MATHEMATICS
AND COMPUTERS

TERMTRAN

A Programming System for
Engineering Calculations
Via Remote Terminals

Authors:

G. B. Knight

R. E. Neu

LEGAL NOTICE

This report was prepared as an account of Government sponsored work. Neither the United States, nor the Commission, nor any person acting on behalf of the Commission:

A. Makes any warranty or representation, expressed or implied, with respect to the accuracy, completeness, or usefulness of the information contained in this report, or that the use of any information, apparatus, method, or process disclosed in this report may not infringe privately owned rights; or

B. Assumes any liabilities with respect to the use of, or for damages resulting from the use of any information, apparatus, method, or process disclosed in this report.

As used in the above, "person acting on behalf of the Commission" includes any employee or contractor of the Commission, or employee of such contractor, to the extent that such employee or contractor of the Commission, or employee of such contractor prepares, disseminates, or provides access to, any information pursuant to his employment or contract with the Commission, or his employment with such contractor.

UNION CARBIDE CORPORATION
NUCLEAR DIVISION
Computing Technology Center
Oak Ridge, Tennessee

**THIS PAGE
WAS INTENTIONALLY
LEFT BLANK**

Subject Category: MATHEMATICS
AND COMPUTERS

Title: TERMTRAN: A Programming
System for Engineering
Calculations Via Remote
Terminals

Authors: G. B. Knight
R. E. Neu

ABSTRACT

This report describes a programming system for performing a variety of engineering and scientific calculations using a remote terminal. The system is designed to be fast, economical and easy-to-learn yet still offer the basic essentials of a programming language. The unique feature of the system is its ability to function in a multi-programming environment (OS/360) while offering a minimum of interference to other executing tasks.

The system has been implemented on the IBM 360/50 and has performed very satisfactorily to date. It is available to any user who has access to the Computing Technology Center's teleprocessing network.

THIS PAGE
WAS INTENTIONALLY
LEFT BLANK

ACKNOWLEDGEMENTS

The authors wish to acknowledge Mr. E. C. Witt for his support and pertinent suggestions offered during preparation of the manuscript, and Miss Marsha Willien for her untiring efforts in preparing this report for publication.

THIS PAGE
WAS INTENTIONALLY
LEFT BLANK

TABLE OF CONTENTS

	PAGE
ABSTRACT	3
I. INTRODUCTION	9
II. THE TERMTRAN LANGUAGE.	11
A. STATEMENTS	11
B. CONSTANTS.	12
C. VARIABLES.	13
D. EXPRESSIONS.	14
E. STORAGE ARRANGEMENT OF VARIABLES	17
F. SUBSCRIPTS	18
G. ASSIGNMENT STATEMENT	19
H. CONTROL STATEMENTS	20
I. END STATEMENT.	22
J. INPUT/OUTPUT STATEMENTS.	22
K. FUNCTIONS.	24
L. PROCEDURES	26
III. PROGRAM DIAGNOSTICS.	29
IV. PROGRAM ENVIRONMENT AND LIMITATIONS.	33
V. PROGRAM MODIFICATION AND STORAGE PROCEDURES.	35
VI. REMOTE TERMINAL OPERATING PROCEDURES AND FACILITIES.	39
VII. APPENDIX A TERMINAL IDENTIFICATION CODES.	43
VIII. APPENDIX B ERROR MESSAGES	44
IX. EXAMPLES	47

THIS PAGE
WAS INTENTIONALLY
LEFT BLANK

I. INTRODUCTION

The TERMTRAN (TERMinal TRANslator) system has been developed at CTC to serve a dual purpose. Initially it offered a substantial "in shop" application with which to test the general purpose teleprocessing support being designed and programmed at CTC. Success with this initial purpose implied the more extensive purpose of providing remote demand processing for engineering and technical calculations with the Operating System/360 in use at CTC.

The TERMTRAN compiler language described in this report will be seen to resemble FORTRAN. Although considerable modifications to an operating TERMTRAN program may be required before it can be run in FORTRAN, many of the arithmetic statements can be carried over unmodified. Differences between the TERMTRAN language and FORTRAN resulted from requirements of a relatively simple and fast compiler and from efforts to ease the learning and use of the language by operators for whom computing is only an aid to their primary activity.

In addition to the compiler language provided by TERMTRAN, there are operations providing for line-at-a-time modifications to a program and for maintaining and retrieving from a file of programs. These operations are quite unlike any capability offered or needed by FORTRAN.

The TERMTRAN system has (as of October, 1967) enjoyed considerable success during testing but has not been subjected to field use. It is to be expected that field usage will uncover errors and shortcomings of the system. These will be given attention commensurate with the acceptance and utilization of the system.

Since a terminal using the TERMTRAN system shares the System/360 at CTC with all other terminals and with a background batch processor, one operating such a terminal can expect to experience occasional delays of as much as several minutes before his calculated results are returned to him after a complete job request has been entered. As no field experience on the system is currently available, the frequency or extent of these delays is unknown. With the expected build-up of remote terminal activity, the delays can be expected to increase both in frequency and in duration. For these reasons and because of the rather extensive commitment by CTC to other remote terminal applications, use of the TERMTRAN system by more than a quite limited number of remote stations cannot be undertaken initially.

II. THE TERMTRAN LANGUAGE

A. STATEMENTS

A TERMTRAN program consists of a set of statements which generate machine instructions. These statements may be divided into three primary categories, computational, control and input/output. A computational statement consists of a set of operators applied to constants and/or variables. Execution of a computational statement causes the value of a designated variable to be replaced by the results of the specified operations. Control statements enable the user to alter the usual sequential flow and terminate execution of his program. Input/output statements provide the means for entering and extracting data from a program.

STATEMENT FORMAT

TERMTRAN source statements are relatively free-form with embedded blanks being ignored in most cases. The exceptions to this free-form style are as follows:

1. If a statement is named, the name must start in column 1 and be followed by at least one blank character. A statement name is any combination of 1 to 4 characters with the first character alphanumeric.
2. If column 1 contains the character "C", the statement will be treated as a program comment card.
3. If a statement is not named, column 1 must be blank.
4. Source statements cannot exceed 72 characters in length.
5. Input data definitions cannot exceed 80 characters per line.

B. CONSTANTS

A constant is a fixed, unvarying, numerical quantity. There are two types of numerical constants, integer and real.

INTEGER CONSTANT - a whole number with no decimal point. Integer constants may be positive, negative or zero, signed or unsigned.

Integer constants must be less than eight decimal digits in length.

If the constant is unsigned, it is assumed to be positive.

EXAMPLES

0

25

-33

7777777

INVALID INTEGER CONSTANTS

0.0 (contains a decimal point)

-19. (contains a decimal point)

4,076 (embedded comma)

888888888 (exceeds allowable length)

REAL CONSTANT - a number with a decimal point optionally followed by a decimal exponent. This exponent is written as the letter E followed by a signed or unsigned, one or two digit integer constant. Real constants may be from 1 through 7 decimal digits with a decimal point. A real constant may be positive, negative or zero, signed or unsigned (if unsigned, it is assumed to be positive). The magnitude of the decimal exponent, if used, must be less than 75.

EXAMPLES

1.2

-233.7966

1.7E-21

 (1.7×10^{-21})

-3.2E+04

 $(-3.2 \times 10^4 = -32000.0)$

+17.5E01

(175.0)

INVALID REAL CONSTANTS

21.766345

(exceeds allowable length)

0

(missing decimal point)

1,274.33

(embedded comma)

2.7E-99

(exponent out of range)

1.5E002

(exponent more than two digits)

-7.5E

(exponent missing)

C. VARIABLES

A variable is a symbolic representation of a quantity which can assume a value. The value of the variable may change within the program or remain constant. TERMTRAN variables are two characters in length. The first character of the variable must be alphabetic and the second numeric. Any pair of alphabetic-numeric characters may be used as a variable with the exception of alphabetic O. Variables which start with the letter O are illegal.

VARIABLE TYPES

In order to represent data truthfully, the variable type must correspond to the data type.

Hence, variables may be either integer or real. The variable type is implicitly specified by the first character of the variable. Integer variables must start with the letters I or J. All other variable names are considered real variables with the exception of O which is illegal in any case.

EXAMPLES

I0	(integer variable)
J9	(integer variable)
A1	(real variable)
K5	(real variable)
Z9	(real variable)
Q4	(real variable)

INVALID VARIABLE NAMES

I11	(illegal variable length)
J	(illegal variable length, no numeric character)
O5	(illegal variable name, O is illegal alphabetic)
AB	(second character not numeric)
AA7	(illegal variable length)
#4	(first character not alphabetic)

D. EXPRESSIONS

An expression consists of a set of operations applied to variables and/or constants. The mode of an expression may be integer or

real depending on the type of variables and/or constants involved in the expression. The valid operators are as follows:

OPERATOR	DEFINITION
$\&$	Exponentiation
$*$	Multiplication
$/$	Division
$+$	Addition
$-$	Subtraction

RULES FOR CONSTRUCTING EXPRESSIONS

1. In an expression, the operations must be explicitly defined using one of the five permissible operators. For example, to multiply A1 by B1 the operator $*$ is used:

$$A1 * B1.$$

The multiplication will not be performed if the expression is written:

$$A1 \times B1 \text{ or } A1B1 \text{ or } A1 \cdot B1.$$

2. No two arithmetic operators may appear in a sequence in the same expression. Thus, $A1*-A2$ is illegal. The expression may be written as follows:

$$A1*(-A2) \text{ or } -A2*A1.$$

3. The exponentiation operator, $\&$, should be the only operation defined in an expression. Hence, the exponent must be an unsigned constant or variable. However, it is possible for the base to be any arithmetic expression not involving exponentiation providing it is enclosed in parentheses. For example:

$A1 \& 2$	yields $A1^2$
$(I1 + J2 * J3) \& I0$	yields $[I1 + (J2 * J3)]^{I0}$
$(A1 * B2 / C3) \& A0$	yields $[(A1 * B2) / C3]^{A0}$
$K7 + K8 \& 3 - K9$	illegal position of $\&$
$(A9 + B0) \& (I1 + 1)$	illegal exponent

The mode of an exponential expression is determined by the variable type of the base. Exponentiation is not permitted when the base is integer and the exponent real.

EXPRESSION	MODE
$A1 \& 2$	REAL
$A1 \& A2$	REAL
$I1 \& A2$	ILLEGAL
$I1 \& J3$	INTEGER
$A1 \& 3.5$	REAL
$J5 \& 3$	INTEGER

ORDER OF COMPUTATION

Where parentheses are omitted calculations take place from left to right within a computational hierarchy. Multiplication and division are effectively performed first, then addition and subtraction. When operations with the same hierarchy level are encountered in succession, the calculation proceeds in a left to right manner. For example, the expression:

$$A1 * B1 + D1 * E1 / F5 - K5$$

is effectively evaluated in the following manner:

1. $A1*B1$	Call this result X
2. $D1*E1/F5$	Call this result Y
3. $X + Y - K5$	Final operation.

PARENTHESES. Parentheses may be used to specify the order in which the arithmetic operations are to be performed. Parenthesized expressions will be evaluated before the result is used. Hence, if the above example is written:

$$(A1*B1+D1*E1)/(F5-K5)$$

it will be effectively evaluated in the following manner:

1. $A1*B1$	Call this result X1
2. $D1*E1$	Call this result X2
3. $X1+X2$	Call this result Y1
4. $F5-K5$	Call this result Y2
5. $Y1/Y2$	Final operation.

INTEGER DIVISION: When division of two integers is performed the result is a truncated integer. Thus $5/2$ yields 2 after truncation; $-5/2$ yields -2.

E. STORAGE ARRANGEMENT OF VARIABLES

The variable set ($A0-A9, \dots, N0-N9, P1 \dots Z9$) is placed in ascending storage locations in the following sequence:

I0-I9

J0-J9

A0-H9

K0-N9

P1-Z9

This arrangement allows references to any member of the variable set as a subscripted integer variable. Similarly, any real variable may be referenced by $A0(I)$ where $I=0,1,2,\dots,229$.

F. SUBSCRIPTS

A subscript is an integer used to refer to a particular variable within the variable set. Subscripts must be enclosed in parentheses and be non-negative. Only one subscript may be used, that is, multiple-subscripted variables are not permitted. The following rules apply to the construction of subscripts:

1. Subscripts may contain expressions which employ any of the operators $+$, $-$, $*$, or $/$.
2. Subscripts may contain subscripted integer variables.
3. The result of a subscript must always be non-negative and not greater than the size of the variable set.

EXAMPLES

SUBSCRIPTED VARIABLE	STORAGE LOCATION
$A0(1)$	A1
$I0(0)$	I0
$I0(10)$	J0
$I0(2*10)$	A0
$B5(I5*I7/2+44)$	
$Z1(J5-3)$	
$K3(I2(4)*J2(I4+5))$	

INVALID SUBSCRIPTED VARIABLES

A0(-5)	(negative subscript value)
B3(23000)	(subscript too large)
C4(I0&3)	(illegal operator in subscript).
C(75)	(variable name incorrect)
R3(A2)	(subscript must be an integer)

G. ASSIGNMENT STATEMENT

GENERAL FORM: $a = b$

Where: a is any subscripted or non-subscripted variable.

b is any constant, variable or expression.

The execution of an assignment statement causes the value of the quantity on the left of the equal sign to be replaced by the results obtained from evaluation of the quantity on the right. The mode of the expression on the right of the equal sign must be consistent within itself.

EXAMPLES

$A1 = 5.0 + B3(I5 + 1)$

$A1 = A1 + 1.0$

$Z8 = 27. \&A3$

$K9 = V4 + V5 * 2.1 / K9$

$M5 = L7 + .001$

$I7(I1 * I2 / I3) = I1 / J1 - J0$

$R4 = A7 * 2 + D5$ (illegal, mode not consistent)

If the mode of the variable on the left of the equal sign differs from the mode of the expression on the right, a conversion will be made before replacement occurs. For example, the statement

$$I7 = P5 + .001$$

is processed in the following manner. The value of P5 is increased by .001 and the result truncated and converted to an integer. This integer then replaces the value of I7.

H. CONTROL STATEMENTS

Normally, statements are executed in a sequential manner. To alter this statement by statement sequencing, control statements are used.

GO TO STATEMENT

GENERAL FORM: GO TO XXXX

Where: XXXX is the name of a statement.

When the GO TO statement is executed, control passes to the statement specified. At that point normal instruction sequencing continues. Any statement which immediately follows a control statement should be named, otherwise the statement will never be executed.

EXAMPLE

I0=0

INDX I0=I0+1

J0=J0*I0

LOOP GO TO INDX

Each time the statement named LOOP is executed, control is transferred to the statement named INDX.

IF STATEMENT

GENERAL FORM: IF(a) name1, name2, name3

Where: a is any expression

name1, name2, name3 are statement names.

This statement causes control to pass to the statement named name1, name2, or name3 when the value of the expression (a) is less than, equal to, or greater than zero respectively.

EXAMPLE

```

      IF(I1+J1(2)-4*I2)10,20,30
      :
10   I1=I1&2
      :
20   J1=J1&I1
      :
30   I2=(J2/I1)&2
      :

```

In the above example if the value of the expression $(I1+J1(2)-4*I2)$ is negative, statement named 10 is executed next. If the value of the expression is zero, statement named 20 is executed next. If the value of the expression is positive, statement named 30 is executed next.

NOTE: The exponentiation operator may be used in the IF statement; however, it is subject to the limitations described under RULES FOR CONSTRUCTING EXPRESSIONS.

I. END STATEMENT

GENERAL FORM: END

The END statement serves a dual purpose in the TERMTRAN language. It defines the end of the source program and starting point for program data. In addition, it generates the machine instructions necessary for normal program termination. Hence, the END statement should be the last statement in any program and may appear only once in the program. The END statement can be named so that control may pass to it by a control statement.

J. INPUT/OUTPUT STATEMENTS

The input/output statements enable the user to enter data for his program and display the results of the program execution.

READ STATEMENT

GENERAL FORM: READ

The READ statement is a data directed command. That is, the variable as well as its value is defined by the data format. A single READ statement may define one data item or several data items, depending on the data format. Data items are delimited by a comma, with a slash mark (/) following the last data item to be defined by the current READ statement. READ statements may appear anywhere in a source program, however, it is imperative that the data and the READ statements be synchronized. Data are entered immediately following the END statement. The general form of the data is as follows:

$$VN_1 = V_1, VN_2 = V_2, \dots, VN_n = V_n/$$

Where: VN_i is any non-subscripted variable name.

V_i is the value which is to be assigned to the respective variable name.

EXAMPLE

A1=4.0/

A2=3.1416, C7=1.06E-23, J7=7, X4=0.707/

I5=1/

Q9=27/

In the last example the variable type is real while its value is integer. To facilitate ease of input, this inconsistency is permitted. The integer will be converted to real and assigned to the appropriate variable. However, the reverse is not true. The data:

I5=7.4/

will result in a data diagnostic and the variable will not be assigned a value.

If a READ statement is executed with no data present, the program execution will be terminated. Therefore, to insure proper program execution the READ statements and the input data format must be in agreement.

WRITE STATEMENTS

WRITE DATA

GENERAL FORM: WRITE VN_1, VN_2, VN_3, VN_4

Where: VN_i are non-subscripted variable names.

Execution of the WRITE statement causes the values assigned to the specified variables to be printed. A maximum of four variables may be displayed by a single WRITE command. The variable type implicitly specifies the output format. That is, if the first letter of the variable is I or J the value is printed in integer form; otherwise, the value appears with a decimal point and exponent if required.

EXAMPLE

```
WRITE A1, J7, K3
```

```
WRITE Z5
```

```
WRITE B7, B9, E3, I2
```

```
WRITE HEADING
```

GENERAL FORM: WRITE 'heading information'

Execution of this form of the WRITE command causes the information contained within the apostrophes to be printed. A maximum of 32 characters may be displayed by this statement.

EXAMPLE

```
WRITE 'START OF TEST'
```

```
WRITE 'N      N!'
```

```
WRITE 'GAMMA FUNCTION CALCULATION'
```

K. FUNCTIONS

A number of standard mathematical functions are provided. These include trigonometric, exponential, logarithmic, square root and absolute value functions. These functions are assumed to be of type

real with real arguments. Therefore, they must only be used in real expressions. The available functions are:

SINF (<u>a</u>)	SINE a
COSF (<u>a</u>)	COSINE a
ATAN (<u>a</u>)	ARCTANGENT a
LOGF (<u>a</u>)	$\text{LOG}_e a$
EXPF (<u>a</u>)	e^a
SQRT (<u>a</u>)	$\sqrt{a}$
ABSF (<u>a</u>)	Absolute value of a

The argument a is any valid real expression.

EXAMPLE

A1 = COSF(3.1416/2.0)

Z7=Q3+P0*ATAN(Z7&I3)

L9=SQRT(ABSF(SINF(-3.1416/2.0+A0)))+L8

The last example illustrates the concept of grouping or nesting functions. The value contained in the innermost set of parentheses is evaluated first, then the next level and so forth. Finally, the addition of the variable L8 completes the calculations necessary to define L9.

INVALID FUNCTION REFERENCES

A1=SINE(A3)	(function name incorrect)
J7=ABSF(J3*J7)	(function argument is type integer)
M9=31.0+EXPF(3.9*2)	(function argument is a mixed-mode expression)

P3=LOGF(-1.735) (function argument is mathematically
invalid)

In the last example, although the function is syntactically correct, it is mathematically improper. When the expression involving the function is evaluated the erroneous function argument will cause termination of the program. Thus the user must insure that all function arguments are mathematically and syntactically valid.

L. PROCEDURES

A procedure is a set of source statements which need to be executed frequently and at different places within the source program. To alleviate the needless duplication of these frequently used instructions, the facility has been included to execute the desired instruction set from various locations within the source program. The two statements which accomplish this action are CALL and RETURN.

CALL STATEMENT:

GENERAL FORM: CALL XXXX

Where: XXXX is the name of the first statement in the procedure.

RETURN STATEMENT:

GENERAL FORM: RETURN

To utilize the procedure feature, the user assembles a set of TERMTRAN source statements to accomplish the desired result. The first statement of the procedure must be named and the last statement of the procedure must be the RETURN statement. When the CALL statement in the source program is executed, control passes to the first statement of the procedure in a non-sequential manner. When the

RETURN statement in the procedure is executed control returns to the statement immediately following the CALL statement. It should be noted that any variable needed by the procedure must be set by the source program before the procedure is given control. The CALL statement must not be used within a procedure, since a procedure cannot call another procedure.

EXAMPLE

```

1.0  CALL PRC1
      :
      CALL PRC2
      :
      CALL PRC3
      :
      CALL PRC2
      :
      CALL PRC3
      :
      CALL PRC2
      :
      CALL PRC3

      IF(I0)1.0,HALT,1.0

PRC1  READ
      RETURN

PRC2  A5=SQRT(A3(I0))
      RETURN

PRC3  WRITE A5,I0,Z7
      RETURN

HALT  WRITE 'END OF PROGRAM'

      END

```

In the above example, three different procedures are used. The first reads the data, the second performs a calculation, while the third prints the results of the calculations. Any additional calculations required are performed in a sequential manner outside of the procedures. When the variable I0 is equal to zero the program is terminated. The user must insure that a procedure is never entered sequentially. If this occurs, the results are unpredictable. Hence, the statement immediately preceding a procedure must be a control statement. Also a procedure must not be entered by a control statement other than CALL.

III. PROGRAM DIAGNOSTICS

A number of program diagnostics are available to aid the TERMTRAN user. Each source statement is scanned for proper syntax, parenthesis balance and overall validity. If errors are found, an error code indicating the nature of the discrepancy is printed. In most cases, the statement containing the error and its line number in the program are also printed. A list of the error codes and their definitions follow:

ERROR NUMBER	CAUSE
00	Exceeded core storage bounds*
01	Duplicate statement name
02	Statement name not followed by a blank character
03	Mixed-mode in an expression
04	Statement contains an illegal vocabulary word
05	Undefined transfer point
06	Illegal symbol in an expression
07	Invalid argument mode in a function
08	Illegal syntax in an expression
09	Number of permissible constants has been exceeded*
10	Illegal subscript

11 Parentheses do not balance

13 Illegal exponentiation

* This type of error is caused by program limitations. See the section "PROGRAM ENVIRONMENT AND LIMITATIONS".

The errors described above are compiler oriented. That is, they occur while the source statement is being scanned and interpreted. The occurrence of this type of error results in deletion of source program execution.

Catastrophic errors which occur at execution time will cause program termination and may generate error messages. The error comments and their probable causes are given below:

MESSAGE

CAUSE

"END OF DATA ENCOUNTERED BY READ" This pair of error messages indicate
"*****EXECUTION TERMINATED*****" that a read command was executed with
no data present. It may signal an
expected termination of execution.

"DATA DIAGNOSTIC, SCAN POINTER XX" This message indicates an error in the input data format. XX is a character count which indicates the general location of the error in the input line. The line in error is also printed below this error message.

"ERROR-INVALID LOG
 Sqrt ARGUMENT. EXECUTION TERMINATED"
 COSF
 SIN F This message occurs when the argument
 EXPF of the indicated function is invalid.

"EXPONENTIATION ERROR. RESULTS SET TO 0."

This message occurs with the exponentiation operator when the base is a negative, real entity and the exponent is real. This message does not result in program termination; however, the value of the variable as noted is set to zero.

"*****EXECUTION TERMINATED*****" This error message occurs for a number of different execution errors. These include:

- a) negative subscript
- b) subscript value greater than data area*
- c) invalid read operation
- d) excessive number of program transfers*

*See the section entitled "PROGRAM ENVIRONMENT AND LIMITATIONS".

It should be noted that in the event of a data error, the program continues its execution. Any data that were correctly entered will be

used in subsequent calculations; however, if data errors did exist the variables to be defined by the erroneous data retain their previous value and processing continues. This procedure enables the program to continue its execution and process subsequent valid data.

IV. PROGRAM ENVIRONMENT AND LIMITATIONS

The TERMTRAN system is designed to operate within a teleprocessing environment. In addition to fulfilling the requirements of the teleprocessing tasks, the computer must also serve as a batch processor simultaneously. To insure the harmonious sharing of the computer's facilities by the batch processor and teleprocessing utilities, several restrictions were imposed on the TERMTRAN system. The basic requirements were that the system be fast and economical in its storage requirements. These ends were accomplished by limiting the size of source language programs, minimizing compiler table requirements and providing only minimum diagnostic treatment. Thus a source language program with its data cannot exceed 300 lines and in some cases must be smaller (the critical factor being the number of machine instructions generated from the source statements), a maximum of 100 constants may be used in any program. Violation of these restrictions will result in error codes 00 or 09, respectively. Since the variable area is of fixed length, an attempt to subscript outside this area will result in program termination and an execution time error message. If a source program were to execute an infinite input-output or transfer loop, batch processing would be discontinued; hence, this activity must be disallowed. To prevent the infinite loop situation from occurring, controls on read, write and transfer statements are present. In particular, the following restrictions exist:

- a) A maximum of 50 lines output per source program.
- b) A maximum of 1000 lines of input per source program.
- c) A maximum of 1000 transfers per source program.

The transfer limit is reset to 1000 at the execution of each READ statement. Therefore, an iterative procedure involving more than 1000 iterations can only be accomplished by execution of a read command per 1000 iterations. These controls should allow the batch processor to operate relatively unimpeded and in conjunction with the teleprocessing TERMTRAN system. In addition, a great deal of protection is afforded the TERMTRAN user. He can expect fairly rapid response and prompt notification of source program errors, whether they be compile or execution time errors.

V. PROGRAM MODIFICATION AND STORAGE PROCEDURES

TERMTRAN control functions are provided by a set of fourteen commands, each entered as an asterisk and three letter mnemonic code followed by such additional information as the particular command requires. These functions allow for entering a program and its data, requesting a compilation of a program and editing functions by which the operator may obtain information about the program and its data and modify them. Also provided is the capability for storing and retrieving programs and data under names established by the terminal operator, and deleting and renaming programs so stored. Thus an operator can enter, test, correct and obtain results from a program, then store it. At some later time he can retrieve the program, provide new data for it, with or without modifying the program itself, and re-request its execution. If a program is modified, the modified program may be stored as a different program under a different name, it can replace the unmodified version for subsequent reference, or it can be discarded.

The first line supplied to TERMTRAN must be one of these control function commands. Lines entered before a control function command will be discarded with the message "NOT IN EXPECTED CONTEXT" returned to the terminal.

All of the TERMTRAN control function commands start in column 1 and must contain no embedded blanks. Commands for entering, editing and executing programs are as follows:

- *INI INItiates the program area. It is expected that TERMTRAN language statements will follow and that they will be entered, starting with line 1.
- *COM COMpiles the program in the program area after having stored it in the scratch area for subsequent reference. If the compilation is unsuccessful, diagnostic messages will be returned to the terminal. If it is successful, execution will be attempted and the results of the execution will be returned to the remote terminal.
- *RPL,nnn RePLaces the content of line nnn (three digits specified) by the information immediately following the third digit. Thus, *RPL,005ONE A3 = B3&R5 puts into line five a statement named "ONE" to calculate $B3^{R5}$ and put the result in A3. NOTE: A space between the "5" and "ONE" would cause the name to be erroneously displaced into a location after the first one.
- *DEL,nnn,mmm DELEtes lines nnn through mmm. The line which had been at mmm+1 replaces the line which was at nnn, and lines following line mmm+1 are moved into corresponding lines after nnn.
- *INS,nnn,mmm INSerts lines nnn through mmm into the program where the inserted lines are those immediately following this command. If fewer than mmm-nnn+1 lines follow the command,

blank lines will be left in the program. If too many lines are entered, the "extra" ones will be placed at the end of the program. All lines which were at line nnn and beyond will be moved to make room for the lines to be inserted.

- *SRT,nnn StaRTs entry of subsequent lines at line number nnn and continues sequentially. No old lines from nnn on in the program are retained. NOTE: *SRT,001 is identical in its effect with *INI except that in the latter case, no reference is made to the retained "old" program.
- *EDL,nnn,mmm EDit Lists the program from line nnn through line mmm. The edit listing includes with each line the number by which that line is to be referenced.
- *LST,nnn,mmm LiSTs the program from line nnn through line mmm. Line numbers are not included with this type list.

The commands above are concerned with building, editing, compiling and executing programs. Those which follow have to do with the permanent retention of programs and management of the file in which programs are retained. Programs in the file are identified by names consisting of up to eight characters, none of which may be a comma or an embedded blank.

- *CAT lists the CATalogue of stored programs, by name.

- *STO,job-name STOrs the current program with its data under the name specified by "job-name" provided that name does not conflict with another stored program name.
- *STM,job-name STores a Modified version of the program named "job-name".
If a program of the specified name is in the file it will be replaced by the new one, otherwise the command is ignored.
- *REN,job-name-1, job-name-2 RENames the program having "job-name-1" with "job-name-2". The program is otherwise unchanged, but it may be referenced subsequently only by the new name.
- *FET,job-name FETches the named program and the data stored with it from the program file and makes it available for program editing, compiling and executing.
- *SCR,job-name SCRatches the specified job-name from the file and makes that space available for retention of another program.

VI. REMOTE TERMINAL OPERATING PROCEDURES AND FACILITIES

In order to effectively utilize the features of the remote TERMTRAN system, it is necessary to become familiar with the operating techniques for remote teleprocessing terminals. The TERMTRAN user will undoubtedly be using a typewriter remote terminal. In order to gain access to the computer, a BID or REQUEST key is usually provided. Activating the key indicates that the terminal is ready to send data. When the computer recognizes the request for service by the remote terminal, the keyboard is unlocked and transmission may begin. Following the transmission of the EOT (end of transmission) character, the keyboard is disabled and the remote terminal is logically disconnected from the computer. In order to enter subsequent data or to evoke a program, the BID or REQUEST key must be reactivated.

On this type of device, the carriage return and EOB (end of block) keys are used to indicate the logical end of an input line. If the terminal does not have an EOB character, the carriage return alone serves as the end-of-line indicator. The EOT (end of transmission) key acts as the end of input character. The input lines, source language and data, are queued on a direct access device until the TERMTRAN program is evoked. At that time the source language is processed, a line at a time, until the END statement is encountered. All input lines following the END statement are treated as data input to the compiled source language program. Facilities are available to have this input queue cleared with or without a list of the contents. This should be done routinely before

any source statements are entered from the terminal. This will insure that no residual input is left in the queue. Since the intent is that the teleprocessing system handle a variety of computer applications, including TERMTRAN, it is expected that frequently the user may experience a short delay before his input is processed. To assist the user in determining the status of his request, a feature has been included to indicate the status of the teleprocessing system. The program now being executed, the terminal which evoked that program and the number of programs awaiting execution are reported. By requesting this information the amount of teleprocessing activity can be determined at any time. The format for requesting the above mentioned facilities are as follows:

TERMINAL INPUT

ACTION TAKEN

\$CLRQUEUE

The input queue is cleared of all data and a line count of the contents is printed in the following form:

"QUE IS EMPTY AFTER XXXX RECORDS".

Where: XXXX is the line count.

\$LSTQUEUE

Same as \$CLRQUEUE except that each line is printed as it is cleared.

\$TERMTRAN

Evokes the TERMTRAN program.

/TTT STAT

The status of the teleprocessing system is printed on terminal TTT, in the following form:

"JOB XXXXXXXX FROM YYY NOW EXECUTING.

JOB QUEUE HAS NNN JOBS WAITING. MMM"

Where: XXXXXXXX is the program name.

YYY is the remote terminal
name.

NNN is the number of programs
awaiting execution.

MMM is the number of programs
executed by the telepro-
cessing system.

It must be noted that each of the above terminal inputs must terminate with a carriage return, EOB (if available) and EOT. The valid terminal names will be found in Appendix A.

One additional feature, the backspace key, should be pointed out. If an error is detected in an input line prior to issuing the carriage return or EOB for that line, the backspace key may be utilized to correct the error. The procedure involves backspacing to the character in error, retyping that character and all additional characters from the point of error to the end of the line.

In light of the above comments, the general structure of a set of TERMTRAN input to execute a source program is as follows:

[ACTIVATE BID OR REQUEST KEY]

*INI^E_(C O)
R B

Initialize the system.

```

READ (C E
      R O)
      B
:
:
source statements (C E
                  R O)
                  B
:
:
END (C E
     R O)
     B

```

Read in data items.

Define action to be taken on input data.

End of source program.

```

data values / (C E
               R O)
               B
data values / (C E
               R O)
               B
:
:
last data item / (C E
                  R O)
                  B
*COM (C E E
      R O O)
      B T

```

Program data starts here.

EOT character ends the transmission from this terminal.

[ACTIVATE BID OR REQUEST KEY]

```

$TERMTRAN (C E E
            R O O)
            B T

```

Requests the TERMTRAN system to execute the data in the input queue.

```

(C E
 R O) - carriage return, end of block (if available).
      B

```

```

(C E E
 R O O) - carriage return, end of block, end of transmission.
      B T

```

VII. APPENDIX A

TERMINAL IDENTIFICATION CODES

TERMINAL CODE	TERMINAL LOCATION
CTC	Computing Technology Center, Bldg. K-1007, K-25
DTI	Division of Technical Information Ex- change
NSI	Nuclear Safety Information Center, Bldg. 9711-1, Y-12
ORO	Atomic Energy Commission, Oak Ridge Operations
PRC	Plant Records Department, Bldg. K-1001, K-25

VIII. APPENDIX B

ERROR MESSAGES

This appendix contains a list of additional execution time error comments, their probable causes and suggested user response.

MESSAGE	CAUSE
ERROR IN COMPILATION. EXECUTION DELETED	One or more compiler errors occurred. Correct the errors and execute the program again.
PROGRAM NAME NOT IN CATALOG	A request to scratch, rename, modify or store has been given for a non-existent program name. Reissue the request using the correct program name.
PROGRAM DIRECTORY IS FULL	A request to store a program has been given but no space is available on the storage medium. Contact CTC personnel.
PROGRAM NAME ALREADY IN DIRECTORY	A request to rename or store a program has been given with a non-unique program name specified. Alter the program name and re-request the action.

PROGRAM EXCEEDS 300 STATEMENTS	An attempt to store a program exceeding 300 lines has been requested. Modify program length and re-request the action.
XXXX HAS INVALID ACTION CODE	A syntactical error in one of the modification or storage commands was detected. XXXX is the erroneous command entered. Correct the command and re-request the action.
NO PROGRAM IN CORE, HENCE NO ACTION TAKEN	A request for compilation and execution has been given prior to entering or fetching a program.
TOO MANY LINES -- READING SUSPENDED	An attempt to enter more than 300 lines of program statements and data has been made. Modify input and re-request action.
XXXX,NNN HAS INVALID LINE NUMBER--IGNORED	A request to modify a line number not within the program bounds has been made. XXXX is the command, NNN is the line number specified. Correct the line number and re-request the action.

NOT IN EXPECTED CONTEXT

The line printed above this message occurred in the input queue with no preceding control function command. Re-enter data following the appropriate control function command.

PROGRAM INTERRUPT-X. JOB TERMTRAN FROM TTT HAS BEEN TERMINATED.

This message occurs when a catastrophic execution error is found. X is an error code (see table below), TTT is terminal identification code. Correct the errors and execute the program again. If the error persists contact CTC personnel.

INTERRUPT CODE (X)

PROBABLE CAUSE

1	ERRONEOUS TRANSFER POINT
4	ERRONEOUS TRANSFER POINT
5	ERRONEOUS TRANSFER POINT
6	ERRONEOUS TRANSFER POINT
7	INVALID DATA
8	INTEGER OVERFLOW
9	INTEGER DIVIDE ERROR
C	EXPONENT OVERFLOW
D	EXPONENT UNDERFLOW
F	REAL DIVIDE ERROR

IX. EXAMPLES

The following examples serve to illustrate some of the features of the TERMTRAN system. The information which appears in lower case represents input from the remote terminal to the TERMTRAN system. The responses from the TERMTRAN system to the terminal are transmitted in upper case. It should be noted that the transmission control characters, end of block and end of transmission, are non-printing characters; hence they do not appear on the examples. The first example depicts the creation of a program to compute the mean and standard deviation of a set of data, its successful compilation, execution and finally storage on the program retention file.

EXAMPLE 1

```
*ini
init  i0=0
rd1   read  x0, a data point
      if(x0)done,rd2,rd2
rd2   a0(i0)=x0
      i0=i0+1
      go to rd1
done  k1=i0
      i0=i0-1
c     compute mean of the data points
      i1=0
mean  x1=a0(i1)+x1
```

```

        y0=a0(il)&2+y0
        il=il+1
        if(il-i0) mean,mean,10
10      y1= x1&2/k1
        x1=x1/k1
        write ' x1 = average of data points '
        write x1
c
c      compute standard deviation of data '
c
        sl=sqrt(y0-y1)
        write ' sl = standard deviation of data '
        write sl
        go to init
        end

x0=1./
x0=2./
x0=3./
x0=4/
x0=-1/

*com

$termtran

*COM

STORE ON SCRATCH AND EXECUTE TERMTRAN

COMPILATION SUCCESSFUL. EXECUTION

```

```

X1 = AVERAGE OF DATA POINTS
X1=  +2.500000

S1 = STANDARD DEVIATION OF DATA
S1=  +2.236068

END OF DATA ENCOUNTERED BY READ
***** EXECUTION TERMINATED *****

*sto,testpgm

$termtran

*STO,TESTPGM

031 LINES READ FROM SCRATCH.

REQUESTED ACTION PERFORMED.

END OF DATA FROM REMOTE TERMINAL.

```

In the second example a program with its data is retrieved from the program retention file, listed on the remote terminal, and subsequently compiled and executed.

EXAMPLE 2

```

*fet,gbktst01

*1st,001,050

$termtran

REQUESTED ACTION PERFORMED

041 LINES READ FROM SCRATCH.

1      WRITE ' '

      READ A0,B0,I0

```

```

X0 = B0
CALL ANAL
A1 = T0
X0 = A0
CALL ANAL
A1 = A1-T0
WRITE A0, B0, A1
5      W0 = I0
      I1 = 2
      H0 = (B0 -A0)/W0
      X0 = A0
      CALL KERN
      S0 = K0
10     X0 = X0 + H0
      CALL KERN
      S0 = S0 + 4.0*K0
      X0 = X0 + H0
      CALL KERN
      S0 = S0 +K0 +K0
      I1 = I1 + 2
      IF(I1 -10) 10, 20, 20
20     X0 = X0 + H0
      CALL KERN
      S0 = S0 + 4.0*K0
      X0 = X0 + H0

```

```

      CALL KERN

      S0 = (S0 +K0)*H0/3.

      E0 = A1 -S0

      WRITE H0, S0, X0, E0

      GO TO 1

KERN   K0 =(3.*X0 -9.)*X0*X0 +1.

      RETURN

ANAL   X1 = X0&3

      TO = (0.75*X0 -3.0)*X1 +X0

      RETURN

30     END

      A0 = 0.75, B0 = 3.5

      I0 = 22/

I0 = 4

*com

$termtran

*COM

041 LINES READ FROM SCRATCH.

COMPILATION SUCCESSFUL. EXECUTION

A0=      +.750000      B0=   +3.500000      A1= -12.299805

H0=      +.125000      S0=  -12.299805      X0=  +3.500000      E0=  0.0

A0=      +.750000      B0=   +3.500000      A1= -12.299805

H0=      +.687500      S0=  -12.299805      X0=  +3.500000      E0=  0.0

      END OF DATA ENCOUNTERED BY READ

***** EXECUTION TERMINATED *****

```

The final example indicates how a procedure may be retrieved from the program retention file and included in a user program. The first statement of the procedure transfers control to the user at statement name USER. The procedure is then executed by a CALL command.

EXAMPLE 3

```
*edl,001,040
```

```
$termtran
```

```
*EDL,001,040
```

```
055 LINES READ FROM SCRATCH.
```

```
001      GO TO USER
```

```
002 C    GENERAL LAGRANGE INTERPOLATION PROCEDURE.
```

```
003 C    J0 IS THE DEGREE OF THE POLYNOMIAL.
```

```
004 C    F0 IS THE ARRAY OF ARGUMENTS
```

```
005 C    X0 IS THE ARRAY OF INDEPENDENT VARIABLES.
```

```
006 C    I0 IS THE POINTER TO THE FIRST ELEMENTS TO BE USED.
```

```
007 C    V0 IS THE NEW VALUE OF THE INDEPENDENT VARIABLE.
```

```
008 C    W1 GETS THE NEW VALUE OF THE DEPENDENT VARIABLE.
```

```
009 C    W0, I9, I1 and J1 ARE WORKING STORE LOCATIONS.
```

```
010 PLYN  I9 = I0 + J0
```

```
011      W1 = 0.
```

```
012      I1 = I0
```

```
013 PLY1  J1 = I0
```

```
014      W0 = F0(I1)
```

```
015 PLY2  IF(I1 -J1) PLY4, PLY3, PLY4
```



```
016 PLY4  W0 = W0*(V0 -X0(J1))/(X0(I1) -X0(J1))
017 PLY3  IF(I9 -J1) PLY6, PLY6, PLY5
018 PLY5  J1 = J1 +1
019      GO TO PLY2
020 PLY6  W1 = W1 + W0
021      IF(I9 -I1) PLY8, PLY8, PLY7
022 PLY7  I1 = I1 + 1
023      GO TO PLY1
024 PLY8  RETURN
025 USER  READ TO LOAD THE ARRAYS
026 2     READ V0 AND J0
027      J2 = J0/2
028      I2 = 0
029 3     IF(V0 -X0(I2)) 5, 4, 4
030 4     I2 = I2 +1
031      GO TO 3
032 5     I0 = I2 -J2 -1
033      IF( I0) 6, 7, 7
034 6     I0 = 0
035 7     IF(I0 + J0 -30)9, 9, 8
036 8     I0 = 30 -J0
037 9     CALL PLYN
038      WRITE V0,W1,J0,I0
039      GO TO 2
040      END
```

Note: Data for the READ statements follow starting in line 41.