CONTROL DATA
CORPORATION

CONTROL DATA®
STAR COMPUTER SYSTEM

FORTRAN LANGUAGE REFERENCE MANUAL

New features, as well as changes, deletions, and additions to information in this manual are indicated by bars in the margins or by a dot near the page number if the entire page is affected. A bar by the page number indicates pagination rather than content has changed.

REVISION RECORD

REVISION	DESCRIPTION
A	Original printing. This manual supersedes FORTRAN Reference Manual, Publication Number 60384500.
(6-20-74)	
Publication No. 60386200	

Additional copies of this manual may be obtained from the nearest Control Data Corporation sales office.

Address comments concerning this manual to:

CONTROL DATA CORPORATION
Software Documentation
215 MOFFETT PARK DRIVE
SUNNYVALE, CALIFORNIA 94086

or use Comment Sheet in the back of this manual

PREFACE

This manual describes the FORTRAN language for the CONTROL DATA® STAR computer line. STAR FORTRAN is designed in compliance with the guidelines for ANSI FORTRAN (ASA document X3.9-1966), established by the American National Standards Institute. STAR FORTRAN is also designed with extensions to the ANSI FORTRAN capabilities. The extensions provide additional capabilities and make efficient use of the unique architecture of the STAR computers.

The STAR FORTRAN compiler functions under control of the STAR Operating System and is non-conversational. The compiler, object time libraries, and generated object programs are re-entrant and location independent. The compiler provides options for object code optimization, implicit vectorization, source listings, assembly listings, memory maps, and cross reference listings.

The reference section, Part I, contains a full description of the STAR FORTRAN language. Part II contains sample programs designed to illustrate some capabilities of the compiler. Discussions of some programming considerations are included with the sample programs.

For additional information about related software refer to the following document:

STAR Operating System Reference Manual, Publication Number 60384400.

This product is intended for use only as described in this document. Control Data cannot be responsible for the proper functioning of undescribed features or undefined parameters.

CONTENTS

APPENDIXES

FIGURES

CODING FORTRAN STATEMENTS I-1

A FORTRAN program is written on a coding form as illustrated in figure 1-1. If a statement is too long to fit on a 72-character line it may be continued to 19 additional lines. No more than one statement is permitted on a single line. Executable statements specify action the program is to take, and non-executable statements describe characteristics of operands, statement functions, arrangement of data, and format of data. Lines may also be used for comments, which are ignored by the compiler.

Each line on the coding form is a sequence of characters from the following character set.

FORTRAN CHARACTER SET

Alphabetic:	A to Z	
Numeric:	0 to 9	
Special:	b	Blank†
	=	Equals
	+	Plus
	-	Minus or Dash
	*	Asterisk
	/	Slash or Divide
	(	Left parenthesis
	)	Right parenthesis
	,	Comma
	.	Decimal point/period
	$	Currency symbol
	&	Ampersand
	'	Apostrophe
	:	Colon

†Throughout this manual the blank is shown as a b where its presence is significant, otherwise a space is used.

CONTROL DATA FORTRAN CODING FORM

PROGRAM	PASCAL	NAME	
ROUTINE		DATE	PAGE OF

TYPE	STATEMENT NO.	CONT.	FORTRAN STATEMENT (0 = ZERO Ø = ALPHA O; 1 = ONE I = ALPHA I; 2 = TWO Z = ALPHA Z)	SERIAL NUMBER
			PROGRAM PASCAL (OUTPUT)	
			INTEGER L(1:1)	
			DATA L(1:1)/1/	
C				
			PRINT 4, (I, I=1, 1:1)	
	4		FORMAT(14H1COMBINATIONS OF MI THINGS TAKEN NI AT A TIME./1X,I2,3H-IN-I5)	
		$	1:1	
C				
			DO 2 I=1, 10	
			K=1:1-I	
			L(K) =1	
			DO 1 J=K, 10	
	1		L(J) =L(J)+L(J+1)	
	2		PRINT 3, (L(J), J=K, 1:1)	
	3		FORMAT(1X,I5)	
			STOP	
			END	

Figure 1-1. Program PASCAL

FORTRAN STATEMENTS

Column 1	C indicates comment line (not processed by compiler)
Columns 1-5	Numeric statement label; blanks and leading zeros are ignored
Column 6	Any character other than blank or zero denotes continuation of a statement; does not apply to comment line
Columns 7-72	Statement; blanks are ignored except in Hollerith strings
Columns 73-80	Identification field (not processed by compiler)

STATEMENTS AND LABELS

Each statement begins with an initial line which must contain either a blank or the digit 0 in column 6; columns 1 through 5 may be blank or contain a numeric statement label. A given statement label must not be used more than once in the same program unit. The numeric value of a statement label has no significance and any values between 1 and 99999 may be used in any order.

CONTINUATION OF STATEMENTS

Statements are coded in columns 7-72; a statement longer than 66 columns may be continued on as many as 19 lines. A character other than blank or zero in column 6 indicates a continuation.

Columns 1 through 5 are ignored unless column 1 contains a C which makes it a comment line. An END statement cannot be continued.

COMMENTS

A C in column 1 denotes a comment line and, except for being printed in the output listing, the remainder of the line has no significance. A comment line must be followed immediately by an initial line of a statement or by another comment line. No other restrictions are imposed on the placement of comments within a program.

COLUMNS 73-80

Columns 73 through 80 may contain any valid STAR characters; they have no effect on the program. Generally, these columns are used to order punched cards in the deck. Information in these columns is printed with the source listing.

SYMBOLIC NAMES

A symbolic name consists of one to eight alphabetic characters or digits. The first character of a symbolic name must be an alphabetic character. Symbolic names are used to identify: variables, program units, functions and subroutines, common blocks, and namelists.

CONSTANTS

There are three classes of constants – those that deal with numeric values, those that deal with logical values .TRUE. and .FALSE., and those that deal with literal character strings.

INTEGER CONSTANTS

Form

$n_1 n_2 \ldots n_m$

Element Definition

Each n is a decimal digit

Examples

0	3619257
3471	5
775	14669

The maximum value of an integer constant is $2^{47}-1=140737488355327$

An integer constant is a string of digits written without a decimal point. It must not contain embedded commas. Its value is that of the digit string interpreted as a decimal numeral.

REAL CONSTANTS

Form

n.

.n

n.n

n.E±s

.nE±s

n.nE±s
nE±s

Element Definition

n	Each n is a string of decimal digits
.	Denotes itself
E	Denotes itself
±	Is either a plus or minus sign or is omitted to imply plus
s	Is an integer constant

Examples

```
0.    3.97    .55772    1E10    6.024E-23    .59E+5
```

The range of values is zero or .519211284565733E–8617 through .953708115431876E+8645, and the precision retained is approximately 15 significant digits.

A real constant must not contain embedded commas. Its value is that of the decimal number multiplied by ten raised to the power of the constant which follows the E. Thus, the value of 1E10 is ten billion. The exponent E+0 is assumed if the constant contains no E specification.

DOUBLE PRECISION CONSTANTS

A double precision constant is written and interpreted identically to a real constant except that the letter D and integer exponent value must be present in a double precision constant.

Examples

```
0.D0    1.D0    7D5    2D-1    6.023D24
```

The range of values is the same as for real constants, however the precision retained is approximately 30 significant digits.

COMPLEX CONSTANTS

A complex constant is written as a pair of real constants separated by a comma and enclosed in parentheses. The first real constant denotes the value of the real part and the second real constant denotes the value of the imaginary part. Either constant can be preceded by a plus or minus sign. Complex values are represented internally by two consecutive computer words.

Examples

```
(.3,2.)     Has the value .3+2i
(-3E1,0.)   Has the value -30+0i
```

LOGICAL CONSTANTS

There are two logical constants:

Form

```
.TRUE.
.FALSE.
```

Element Definition

The periods are part of the constants.

HOLLERITH CONSTANTS

Form

nHs

Element Definition

n	Integer in the range of 1 to 255
H	Denotes itself
s	String of exactly n characters

Examples

```
5HLABEL
7HMAD DOG
```

Blanks are significant in the Hollerith string. This type of constant can be used as data in a DATA statement, as an argument in a CALL statement, or as a character expression.

CHARACTER CONSTANTS

Character constants can be used wherever Hollerith constants are used.

Form

's'

Element Definition

'	Denotes itself
s	String of 1 to 255 characters

An apostrophe can be represented within the string by two consecutive apostrophes; the additional apostrophe is not counted in the maximum number of characters allowed.

HEXADECIMAL CONSTANTS

Form

Zd

Element Definition

Z	Denotes itself
d	Is a string of hexadecimal digits

The value is normally an integer with d interpreted as a number in base 16 notation (hexadecimal). Hexadecimal digits are 0, 1, 2, 3, 4, 5, 6, 7, 8, 9, A, B, C, D, E, and F. The digit F is equal to decimal 15.

Examples

Z9AE3	Value is 39651
Z100	Value is 256

VARIABLES

In FORTRAN a variable is a symbolic name representing a quantity which may assume different values during program execution. Each variable is associated with a storage location; when the variable is used, it has the value determined by the contents of its location and the type associated with the symbolic name used to identify the variable.

The value of a variable is changed during program execution by:

Executing an assignment statement where its name occurs to the left of the equals sign

Reading a new value into it

Using it as a DO index (including implied DO's in an I/O list)

Using it in an ASSIGN statement

Using it as an argument to a subprogram that changes the argument value

Changing the value of a variable to which it has been equivalenced

Unless overridden by a Type or IMPLICIT statement, the type of a variable is determined by the first character of the variable name. The variable is integer if the first character of its symbolic name is I, J, K, L, M, or N and it is real if the first character of the name is any other letter. This convention is the traditional FORTRAN method of implicitly specifying the type of a variable as being either integer or real.. In this manual this convention on types is assumed unless otherwise noted.

ARRAYS

A set of variables may be thought of collectively as an array and identified by a single array name. A particular element of the array is identified by following the array name with a subscript which specifies the position of the element within the array. The subscript is a list of subscript expressions enclosed in parentheses. One to seven subscript expressions may appear in the list, separated by commas. The size and number of dimensions associated with an array name are declared in a DIMENSION, Type, or COMMON statement. Type is associated with array names in the same manner as with variable names and the type of each element of the array is determined by the array name. The number of elements in an array is the product of the dimensions, and the number of dimensions in the array is indicated by the number of subscripts in the declaration.

Example

```
DIMENSION APE(7,3), LIP(16), TOT(2,2,2,2)
```

APE is a two-dimension array of 21 real elements.

LIP is a one-dimension array of 16 integer elements.

TOT is a four-dimension array of 16 real elements.

The number of subscript expressions used to reference an element of an array must be the same as the number of dimensions in the array declaration, and the value of each expression should be between one and the corresponding value in the declaration.

The entire array may be referenced by the unsubscripted array name when it is passed as an argument to a subprogram or referenced in an input/output list or DATA statement. When the entire array is referenced, the elements are ordered with the value of the first subscript varying through its range, then the second subscript increased by one with the first going through its range again, and so on until each subscript has gone throughout its entire range.

Example

```
C       WHERE DECLARATION WAS  A(3,3,3)
        READ(5,3) A
    3 FORMAT(E10.2)
```

Would read the elements in the following order:

A(1,1,1)
A(2,1,1)
A(3,1,1)
A(1,2,1)
A(2,2,1)
A(3,2,1)
A(1,3,1)
A(2,3,1)
A(3,3,1)
A(1,1,2)
A(2,1,2)
A(3,1,2)
A(1,2,2)
A(2,2,2)
A(3,2,2)
A(1,3,2)
A(2,3,2)
A(3,3,2)
A(1,1,3)
A(2,1,3)
A(3,1,3)
A(1,2,3)
A(2,2,3)
A(3,2,3)
A(1,3,3)
A(2,3,3)
A(3,3,3)

SUBSCRIPTS

A subscript can be any arithmetic expression of type integer, real, or double precision. When the value of the expression is not integer it is truncated to integer.

ARRAY AND SUBARRAY REFERENCES

An array name reference is a subarray reference in which an array name is not qualified by a subscript expression. It identifies all the elements of the array.

Example

```
C      WHERE DECLARATION WAS  A(100,100)
   12 READ(2,102) A
```

A Represents all the elements of the array A in the order in which A is stored internally.

A subarray reference simultaneously identifies one or more array elements. In an implied-DO subarray reference, the array name is qualified by one of the implied DO forms. The basic forms of the implied DO subscript are as follows:

$M_1:M_2:M_3$

$M_1:M_2$

*

$M_1:*:M_3$

$M_1:*$

The M_i are indexing parameters. M_1 is the initial scalar subscript value. M_2 is the terminal scalar value, which may be expressed as * when the value of M_2 is identical to the declared length of the dimension. M_3 is the index incrementation value. M_3 assumes the value 1 when omitted. M_1, M_2, and M_3 must be unsigned integer constants or integer variables.

Example

```
C      WHERE DECLARATION WAS  A(4,4), B(5)
C      A(1:4:2,2:3)       REPRESENTS     A(1,2), A(3,2), A(1,3),
C                                        A(3,3)
C      B(*)               REPRESENTS     B(1),B(2),B(3),B(4),B(5)
C      B(2:*:2)           REPRESENTS     B(2),B(4)
C      B(3:*)             REPRESENTS     B(3),B(4),B(5)
C      A(*,2:3)           REPRESENTS     A(1,2), A(2,2), A(3,2),
C                                        A(4,2), A(1,3), A(2,3),
C                                        A(3,3), A(4,3)
```

The array name reference may be transformed to the equivalent implied DO reference. In fact, the compiler transforms array name references to equivalent implied DO references before processing. The following example illustrates this transformation.

Example

```
C       WHERE DECLARATION WAS  X(10,20,30)
C       ARRAY REFERENCE X IS EQUIVALENT TO X(1:10,1:20,1:30)
```

Expressions are used to specify a computation or a relationship between two or more constants and/or variables. In its simplest form, an expression consists of a single constant or variable. More complex expressions are formed from elements, operators, and parentheses. This section gives the formation and evaluation rules for four types of expressions: arithmetic, relational, character, and logical. Arithmetic expressions have a value whose type is integer, real, double precision, or complex. Logical expressions always have a truth value of true or false. Relational expressions appear within the context of logical expressions and only have values of true and false. Character expressions have values which are character strings.

The formation and evaluation rules in this section conform to ANSI rules but are liberalized with respect to operand types.

ARITHMETIC EXPRESSIONS

An arithmetic expression is a sequence of constants, variables, and function references separated by operators and parentheses. For example, the following arithmetic expression is valid:

```
(A-3.5)*F+C/D**E
```

FORTRAN arithmetic operators:

Operator	Representing
+	Addition
-	Subtraction
*	Multiplication
/	Division
**	Exponentiation

Arithmetic elements can be any of those listed below:

Primary	An arithmetic expression enclosed in parentheses, a constant, a variable reference, an array element reference, or a function reference
Factor	A primary or a construct of the form: primary**primary
Term	A factor or a construct of one of the forms: term/factor or term*term
Signed term	A term immediately preceded by + or -
Simple arithmetic expression	A term or two simple arithmetic expressions separated by + or -
Arithmetic expression	A simple arithmetic expression, a signed term, or either of the preceding forms immediately followed by + or - immediately followed by a simple arithmetic expression

A primary of type double precision, real, or integer may be exponentiated by any of the types double precision, real, or integer as shown in the following table.

A primary of type complex may be raised only to an integer or real factor. Only in these cases is the exponentiation operation defined.

Arithmetic operators other than exponentation may be used to combine any admissible elements of the same type; the resultant element will be the same type. Further, an admissible real element may be combined with an admissible integer, double precision, or complex element; the resultant element will be type real, double precision, or complex, respectively.

Type of Result for ab**

a \ b	Complex	Double Precision	Real	Integer
Complex	Illegal	Illegal	Complex	Complex
Double Precision	Illegal	Double Precision	Double Precision	Double Precision
Real	Illegal	Double Precision	Real	Real
Integer	Illegal	Double Precision	Real	Integer

The expression a**b**c**d is defined to mean a**(b**(c**d))

Type of Result for a*b, a/b,a-b,a+b

a\b	Complex	Double Precision	Real	Integer
Complex	Complex	Complex	Complex	Complex
Double Precision	Complex	Double Precision	Double Precision	Double Precision
Real	Complex	Double Precision	Real	Real
Integer	Complex	Double Precision	Real	Integer

RELATIONAL EXPRESSIONS

A relational expression consists of two arithmetic or character expressions, separated by a relational operator, for which the logical result is true or false. When two character expressions are separated by a relational operator, the comparison proceeds from left to right one character at a time. The hierarchy of characters is determined by the collating sequence of the processor. When two character expressions of differing length are compared, the shorter of the two character expressions is treated as though it were padded with blanks on the right until the expressions are of equal length.

One arithmetic expression may be of type integer, real, or double precision; and the other may be any of the types integer, real, or double precision. Arithmetic expressions that are of unequal type are converted before comparison as follows:

Type Conversion for Relational Arithmetic Expressions a.OP.b

a \ b	Double Precision	Real	Integer	Character Hollerith
Double Precision	Double Precision	Double Precision	Double Precision	Illegal
Real	Double Precision	Real	Real	Illegal
Integer	Double Precision	Real	Integer	Illegal
Character Hollerith	Illegal	Illegal	Illegal	Character

Relational operators:

Operator	Representing
.LT.	Less than
.LE.	Less than or equal to
.EQ.	Equal to
.NE.	Not equal to
.GT.	Greater than
.GE.	Greater than or equal to

Type of Relational Result for a.OP.b

a \ b	Double Precision	Real	Integer	Character/Hollerith
Double Precision	Logical	Logical	Logical	Illegal
Real	Logical	Logical	Logical	Illegal
Integer	Logical	Logical	Logical	Illegal
Character/Hollerith	Illegal	Illegal	Illegal	Logical

LOGICAL EXPRESSIONS

A logical expression is formed with the logical operators and logical elements, listed below, and has a logical value of true or false.

Logical operators:

Operator	Representing
.OR.	Logical disjunction
.AND.	Logical conjunction
.NOT.	Logical negation

The logical elements are listed below:

Logical primary	A logical expression enclosed in parentheses, a relational expression, a logical constant, a logical variable reference, a logical array element reference, or a logical function reference
Logical factor	A logical primary or .NOT. followed by a logical primary
Logical term	A logical factor or a construct of the form: logical term .AND. logical term
Logical expression	A logical term or a construct of the form: logical expression .OR. logical expression

CHARACTER EXPRESSIONS

An expression of type character consists of only one element.

It may be one of the following:

Character constant

Hollerith constant

Character array element reference

Character variable reference

Character function reference

EVALUATION OF EXPRESSIONS

A part of an expression need be evaluated only if necessary to establish the value of the expression.

The rules for formation of expressions imply the binding strength of operators. The range of the subtraction operator is the term that immediately succeeds it. Evaluation may proceed according to any valid formation sequence (except as noted below).

When two elements are combined by an operator, the order of evaluation of the elements is optional. If mathematical use of operator is associative, commutative, or both, full use of these facts may be made to revise orders of combination, provided integrity of parenthesized expressions is not violated. The value of an integer factor or term is the nearest integer whose magnitude does not exceed the magnitude of the mathematical value represented by that factor or term. Since the associative and commutative laws do not apply in the evaluation of integer terms containing division, the evaluation of such terms must proceed from left to right.

Any use of an array element name requires evaluation of its subscript. The evaluation of functions appearing in an expression must not alter the value of any other element within the expressions, assignment statement, or CALL statement containing the function reference. The type of the expression that contains a function reference or subscript does not affect, nor is it affected by, the evaluation of the actual arguments or subscript.

No factor may be evaluated that requires a negative valued primary to be raised to a real or double precision exponent. No factor may be evaluated that requires raising a zero valued primary to a zero valued exponent.

No element may be evaluated whose value is not mathematically defined.

Hierarchy of Operator Evaluation:

Operator	Hierarchy	Type
**	Class 1	Arithmetic
/	Class 2	Arithmetic
*	Class 2	Arithmetic
+	Class 3	Arithmetic
-	Class 3	Arithmetic
.EQ.	Class 4	Relational
.NE.	Class 4	Relational
.GE.	Class 4	Relational
.LE.	Class 4	Relational
.LT.	Class 4	Relational
.GT.	Class 4	Relational
.NOT.	Class 5	Logical
.AND.	Class 6	Logical
.OR.	Class 7	Logical

In an expression with no parentheses or within a pair of parentheses in which unlike classes of operators appear, evaluation proceeds according to the hierarchy of classes listed above.

Where the operators are of the same hierarchical class, evaluation proceeds from right to left for class 1 operators, and from left to right for operators of all other classes.

ASSIGNMENT STATEMENTS I-4

An assignment statement evaluates an expression and assigns this value to a variable or array element. The statement is written in the following form:

Form

var = expr

Element Definition

var Variable or array element name

expr Expression

The meaning of the equals sign differs from the conventional mathematical notation. In FORTRAN it means replace the value of var with the value of expr. The type of stored value is always the type associated with the name to the left of the equals sign. For logical and character expressions it is an error if var is of a type different than expr.

Rules for Assignment var = expr

var \ expr	Integer	Real	D.Precision	Complex
Integer	Assign	Fix and assign	Fix and assign	Fix real part and assign
Real	Float and assign	Assign	Truncate and assign	Take real part and assign
D.Precision	D.Precision float and assign	Extend and assign	Assign	Extend real part and assign
Complex	Float and assign real part; zero imaginary part	Assign real part; zero imaginary part	Truncate and assign real part; zero imaginary part	Assign

Assign	Transmit resulting value expr, without change, to var
Truncate	Convert double precision to real
Extend	Convert real to double precision and fill with zero significance
Float	Convert integer to real
Double precision float	Convert integer to real and extend
Fix	Take real and convert to integer, truncating the fractional part
Real part	Real part of complex expr or var

When var and expr are type character, their lengths can differ. If var is longer than expr, blanks are added on the right until it matches the length of var. If var is shorter than expr, characters are dropped from the right until it matches the length of var, then it is stored.

ARRAY AND SUBARRAY ASSIGNMENT

A multiple value expression produces one or more results. It consists of one or more subarray references and also can contain scalar expressions.

An array expression is evaluated by performing the stated operation on corresponding array elements. Scalar references are treated as arrays of the proper sizes with all elements containing the same value.

Example

```
C       WHERE DECLARATION WAS  X(10,20), Y(10), Z(10,20)
```

The following are array expressions.

X	This expression has 10 by 20 array result.
X+Y(3)	This array expression yields a two dimensional array result. The 200 results produced by adding value of Y(3) to each element of array X.

ARRAY OR MULTIPLE VALUE ASSIGNMENT

The general form of an assignment statement is var = expr

Array assignment occurs when the replaced variable var is a subarray reference. The assignment expression expr may be either a scalar or an array expression. A scalar assignment expression produces one value which is assigned to all identified elements of the referenced array. An array expression must conform to var; identified elements of the subarray var are replaced with the corresponding elements in the array expression results. The conditions of conformability of each subarray reference in expr with the subarray reference var are as follows:

The number of implied DO subscripts of a subarray reference in expr must be exactly equal to that of the subarray reference var.

Each implied DO subscript of a subarray reference in expr must match exactly with an implied DO subscript of the subarray reference var.

Example

```
C       WHERE DECLARATION WAS  X(10,20), Y(10,20), Z(10)
C          SOME LEGAL ARRAY ASSIGNMENT STATEMENTS ARE:
        X=Y+3.0
        X(*,1:20:2)= Y(1:10,1:20:2)
        Y(1:*:2,1)= Z(1:10:2)
C          SOME ILLEGAL ARRAY ASSIGNMENT STATEMENTS ARE:
        X=Z
        X(*:3)= Y(2,*)
        X(1:10:3,2:20:2)= Y(*,*)
```

CONTROL STATEMENTS I-5

Normally, FORTRAN statements are executed sequentially. Control statements are available to alter and control the sequence of execution of statements in the program. Control may be transferred to executable statements; it is an error to reference the statement label of a non-executable statement in a control statement. Control statements are executable and may be referenced by other control statements.

UNCONDITIONAL GO TO STATEMENT

Form

GO TO n

Element Definition

n Statement label of an executable statement in the current program unit

Control is transferred so the statement labeled n is the next statement to be executed.

COMPUTED GO TO STATEMENT

Form

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

Element Definition

n Each n is the label of an executable statement in the current program unit

i Non-subscripted integer variable name

Control is transferred so the statement label n_k is the next statement to be executed, when k is the value of i at execution time. When the value of i is not in the range of 1 to m, the first executable statement following the GO TO will be executed next.

The comma following the right parenthesis and preceding i is optional and may be omitted.

ASSIGN STATEMENT

Form

ASSIGN n TO i

Element Definition

n Statement label of an executable statement in the current program unit

i Non-subscripted integer variable name

This statement is used to put statement label information in i for subsequent use in the execution of an assigned GO TO statement. The label information in i need not be numerically equivalent to the decimal value of n. In fact, i should not be referenced in any statement other than an assigned GO TO until it has been redefined.

ASSIGNED GO TO STATEMENT

Form

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

Element Definition

i Non-subscripted integer variable name

n Each n is the label of an executable statement in the current program unit

At execution time, the most recent definition of i must have appeared in an ASSIGN statement. Control is transferred to the statement label referenced in the most recently executed ASSIGN statement defining i. The comma following i or the comma and the entire list may be omitted. If the list is present, however, the label information in i must match one of the statement labels in this list.

ARITHMETIC IF STATEMENT

Form

IF (expr) n_1,n_2,n_3

Element Definition

expr Arithmetic expression of type integer, real, or double precision

n_1,n_2,n_3 Three executable statement labels in the current program unit

Control transfers to n_1 if the value of expr is negative, n_2 if it is zero, or n_3 if it is positive.

If expr is type real or double precision it is not meaningful to expect a precise zero value, therefore n_2 should be the same as either n_1 or n_2. For example,

```
COS(0.)-1.0
```

is mathematically zero, but in a finite precision computer the value cannot be expected to be close enough to zero to take the n_2 branch in an IF statement.

LOGICAL IF STATEMENT

Form

IF (expr) s

Element Definition

expr Logical expression

s Any executable statement except a DO statement or another logical IF

Example

```
      IF(AMIN1(A,B,C).LE.0.) STOP
      IF(AMAX1(A,B,C).LT.A+B+C-AMAX1(A,B,C)) PRINT 7,A,B,C
    7 FORMAT(3G12.4,29HCOULD BE SIDES OF A TRIANGLE.)
```

If expr is true, s is executed, then the next executable statement following the logical IF is executed. If expr is false, s is not executed; and control goes to the next executable statement following the logical IF.

DO STATEMENT

Form

DO n i = m_1,m_2,m_3

Element Definition

n	Executable statement label in the program unit that physically occurs after the DO statement
i	Non-subscripted integer variable name
m_1,m_2,m_3	Integer constants or non-subscripted integer variable names with values of one or greater

The DO statement is used to execute repeatedly the succeeding statements through the statement label n. The terminal statement with the label n must not be:

GO TO of any form

Arithmetic IF

RETURN, STOP, or PAUSE

Another DO

READ statement containing an ERR= or END= branch

CALL statement which passes a return label

Logical IF that has any of the named executable statements

When value of m_3 is 1, m_3 and the preceding comma may be omitted.

The effect of the DO statement is the same as if the following changes were made:

Replace the DO with the two statements

```
      i = m1
  d   CONTINUE
```

and immediately following the terminal statement insert the three statements

```
      i = i + m3
      IF(i .LE. m2) GO TO d
      i = u
```

where d is a statement label different from any existing label in the program unit, and u is an unknown and unusable integer value.

The preceding definition of the effect of the DO statement is valid for nested DO loops having the same terminal statement under the following conditions. Logical changes must be completed one at a time, starting from the first statement of the program unit and proceeding to the end.

The following rules can be deduced from the above definition of the effect of the DO.

The terminal statement should not be a branching statement.

If a jump is made out of a DO loop, the index variable i has its most recent value.

If the DO is satisfied and control goes to the statement following the terminal statement, the index variable i becomes undefined as a result of efficient implementation.

The range of a DO statement can include other DO statements providing the range of each DO is entirely within the range of the containing DO statement.

If more than one DO loop has the same terminal statement, a transfer to that terminal statement can be made only from within the range of the innermost DO.

When DO loops are nested, each much have different index variables.

The use of, and return from, a subprogram from within a DO loop is permitted.

The following rules are true even though they are not apparent from the preceding definition.

The index variable i and the indexing parameters m_1, m_2, and m_3 cannot be given new values during the execution of the DO loop.

A DO loop can be entered only through the DO statement; however if a transfer has been made from within the range of the DO then a transfer back into the same DO loop is valid if none of the indexing parameters i, m_1, m_2, or m_3 have been redefined.

CONTINUE STATEMENT

Form

CONTINUE

The CONTINUE statement is a dummy executable statement used to carry a statement label. The CONTINUE statement serves no purpose unless it has a label. It is frequently used as the last statement in a DO loop to avoid ending the loop with a branching statement. For example, the following loop, which locates the first non-positive element in a 10-element array, requires the CONTINUE statement.

```
C       WHERE DECLARATION WAS  A(10)
        DO 7 I=1,10
        IF(A(I)) 9,9,7
      7 CONTINUE
C                      -OTHER STATEMENTS-
      9 NPOS=I
```

PAUSE STATEMENT

Form

PAUSE n

Element Definition

n	String of one to five digits or a character constant; n can be omitted

Execution of the PAUSE statement causes program execution to be suspended. The string n will be displayed on the operator's console or at the terminal. If execution is resumed, the program continues with the statement following PAUSE.

STOP STATEMENT

Form

STOP n

Element Definition

n String of one to five digits or a character constant; n can be omitted

Execution of the STOP statement terminates the program and returns control to the operating system. The string n will be displayed on the operator's console or at the terminal.

END STATEMENT

Form

END

The END line indicates to the compiler the end of the program unit. Every program unit must physically terminate with an END line.

RETURN AND CALL STATEMENTS

Technically, these statements may be considered as control statements, but they are discussed with subprograms.

Specification statements are non-executable; they define the type associated with variable and array names, specify the dimensions of arrays, control the sharing of storage, and can assign initial values to variables and elements of arrays.

TYPE STATEMENTS

All variable and array names have an associated type which is implied whenever that name is used. When the programmer does not specify type, it is considered integer if the first character of the name is I, J, K, L, M, or N and real if the first character is any other letter. These defaults for the first character of the name can be overridden by the IMPLICIT statement. The explicit Type statement overrides all others.

IMPLICIT TYPE STATEMENT

The IMPLICIT statement must be the first statement in a main program or it must follow the PROGRAM statement; in a subprogram, it must be the second statement.

Form

IMPLICIT $typ_1(v_1,v_2, \ldots v_m), \ldots typ_k(v_1,v_2, \ldots v_n)$

Element Definitions

typ — Each typ is the name of a variable type:

INTEGER, REAL, DOUBLE PRECISION, COMPLEX, LOGICAL, or CHARACTER†

v — Each v is a single alphabetic character (or two alphabetic characters separated by a minus sign to denote the first and last characters of a range) indicating the initial letters of the variables to be considered type typ

†The word CHARACTER can be followed by *n where n is a decimal number which specifies the element length in bytes. If *n is omitted, the assumed length is one.

The IMPLICIT statement does not alter the type of basic and intrinsic functions; however, in a subprogram, it affects the type of the dummy arguments and the function name, as well as other variables in the subprogram.

Example

The following IMPLICIT statement would alter the default type specifications to make each variable beginning with A through D double precision, each beginning with L logical, and those beginning with Z complex.

```
IMPLICIT DOUBLE PRECISION(A-D), LOGICAL(L), COMPLEX(Z)
```

Explicit typing of specific names with any of the following Type statements overrides IMPLICIT or default typing.

EXPLICIT TYPE STATEMENTS

Form

typ $v_1(k_1)/x_1/,v_2(k_2)/x_2/, \ldots v_n(k_n)/x_n/$

Element Definitions

typ — Name of the Type statement:

INTEGER, REAL, DOUBLE PRECISION, COMPLEX, LOGICAL, or CHARACTER†

v — Each v represents a variable, array, or function name

(k) — Optional; each k represents 1 to 7 integer constants, separated by commas, representing the maximum value of each subscript in the array (in a subprogram they can be integer dummy arguments)

/x/ — Optional; each x represents initial data values as described for the DATA statement

The Type statement is used to override or confirm any implicit typing; this method is preferred for specifying dimension and initial values even though they may be specified in DIMENSION and DATA statements.

The Type statements should occur before the first executable statement in the program unit.

---

†The word CHARACTER can be followed by *n where n is a decimal number specifying the length in bytes of each v . When *n is not specified, the assumed length is 1. If the word CHARACTER is not followed by *n , each v can be followed by its own length specification *n .

DIMENSION STATEMENT

Form

DIMENSION $ar_1(k_1), \ldots ar_n(k_n)$

Element Definitions

ar	Each ar is an array name
k	Each k represents 1 to 7 integer constants, separated by commas, representing the maximum value of each subscript in the array (in a subprogram they can be integer dummy arguments)

The DIMENSION statement is an alternative to the Type statement declaring an array and specifying the size and dimensions. The same name can appear in both a DIMENSION and a Type statement if the (k) is not used in the Type statement.

ADJUSTABLE DIMENSIONS

When an array is passed as a parameter to a subprogram, the array dimension specifications within the subprogram can be integer variables, as well as integer constants, provided the array name and all variable names used for array dimension specifications are dummy arguments of the subprogram. Within the subprogram, dummy arguments representing array names must appear in a DIMENSION or type statement that gives dimension information. If dummy arguments are not dimensioned, they cannot be referenced as an array in the subprogram.

If the dimensions of a dummy array in a subprogram are adjustable, they may change each time the subprogram is called; however, the absolute dimensions of the array must have been declared in a program unit earlier in the calling sequence. The adjustable dimensions can be passed through more than one level of subprogram calls.

Adjustable dimensions cannot be used for arrays that appear in a COMMON statement.

EXTERNAL STATEMENT

Form

EXTERNAL $ext_1, \ldots ext_n$

Element Definition

ext	Each ext is an external procedure name, block data name, or name of an entry point in an external procedure

The EXTERNAL statement declares each ext as a subprogram name rather than a data name. A subprogram name or a basic function name must be declared in an EXTERNAL statement in the calling program unit before it can be used as an argument to another subprogram. When ext is an intrinsic function name, it no longer refers to an intrinsic function within the program unit.

COMMON STATEMENT

Form

COMMON $/lab_1/v_1(k_1),v_2(k_2), \ldots v_n(k_n)//v_1(k_1),v_2(k_2), \ldots v_m(k_m)/lab_j/ \ldots$

or

COMMON $v_1(k_1),v_2(k_2), \ldots v_n(k_n)/lab_1/v_1(k_1),v_2(k_2), \ldots v_m(k_m)/lab_j/ \ldots$

Element Definition

/lab/	Each lab is a symbolic common block name. This name could be the same name as any variable or array name, but they would bear no relationship to each other. Absence of lab denotes blank (unlabeled) common; also, if blank common is the first block in the statement, the slashes can be omitted. The same block name can be used more than once in the same or different COMMON statements within a program unit; in which case, all variables in blocks having the same name will be linked into a single block in order of their occurrence.
v	Each v is a variable or array name
(k)	Optional; each k represents 1 to 7 integer constants, separated by commas, representing the maximum value of each subscript in the array

COMMON is a non-executable statement that allows variables or arrays in a calling program or subprogram to share the same storage locations with variables or arrays in other program units. Variables and arrays are stored in the order in which they appear in the common block specification.

Program units sharing the same common block can assign different names to members of the block; but to identify the same common block, they must use the same block name.

Within subprograms, dummy arguments are not allowed in a COMMON statement.

Dimension information for an array name must be specified only once in a program unit in a Type, COMMON, or DIMENSION statement.

The size of a common block is the maximum storage required for that block in any program unit. The size of a common block in a program unit is the sum of the storage required for all variables and array elements declared in that block, as well as those brought into that block with the EQUIVALENCE statement. Common blocks with the same block name in the various program units that comprise an executable program need not be the same size. Also, the size of blank common in various program units can be different.

Program units can assign the same type to a given position within a common block, determined by the number of storage units from the beginning of the block. In such cases, references to that position reference the same quantity. Except for type character, where each position is one byte, storage units always fall on full word 8-byte boundaries.

EQUIVALENCE STATEMENT

Form

EQUIVALENCE $(grp_1),(grp_2), \ldots (grp_n)$

Element Definiton

grp Each grp is a list of the form:

$v_1,v_2, \ldots v_m$

v Each v is a variable name, array name, or subscripted array name (number of subscripts must be one or must conform to the array declaration)

A single subscript refers to the variable at that position in the array. Elements in an array are ordered as described for unsubscripted array references in section I-2. When an array name is used, it is the same as using the subscript (1).

The EQUIVALENCE statement is a non-executable statement which assigns the elements of grp to the same storage location within the program unit (as opposed to COMMON which assigns variables in different program units to the same location.) When an element of an array is referenced in an EQUIVALENCE statement, the relative locations of the other array elements are also defined. It is incorrect to cause a single storage unit to contain more than one element of the same array.

A logical, integer, or real entity equivalenced to a double precision or complex entity shares the same location as the real or most significant part of the complex or double precision entity. When variables of differing types are equivalenced they share the same location, however, type is associated only with the name used to reference it and that name will determine the interpretation of the item. The comma between the right and left parentheses separating groups can be omitted.

EQUIVALENCE AND COMMON

An element or array is brought into a common block if it is equivalenced to an element in common. Two elements in common, even in different blocks, must not be equivalenced to each other. An array brought into common through EQUIVALENCE can extend the common block beyond the last position; however, an EQUIVALENCE statement is not allowed to extend the origin of a common block.

Example

Given the declarations:

```
COMMON/DESK/ E,F,G
DIMENSION H(4)
```

The following EQUIVALENCE statement is illegal because it would extend the origin of the common block DESK:

```
EQUIVALENCE(E,H(3))
```

but this next statement would be acceptable:

```
EQUIVALENCE(G,H(3))
```

The last EQUIVALENCE implies that E and H(1) share the same locations and F and H(2) share the same location. These statements indicate DESK is four storage units long.

DATA INITIALIZATION STATEMENT

Form

DATA $k_1/x_1/,k_2/x_2/, \ldots k_n/x_n/$

Element Definition

k — Each k is a list of variables, array elements, or arrays. Items in the list are separated by commas. Subscripts used to identify array elements must be integer constants.

x — Each x is a list of constants, optionally signed, any of which can be preceded by the repeat specification j* where j is an integer constant.

The commas after each second slash are optional.

The data statement is non-executable; it assigns initial values to variables or array elements. The rules for initializing values with the DATA statement given here also apply to data initialization with the Type statements described earlier in this section.

The number of items in the data list should be the same as the number of variables in the variable list preceding the data list. Only variables assigned values by a data initialization statement have specified values when program execution begins.

When the form j* appears before a constant, it indicates the number of times the constant is specified. An unsubscripted array name references all elements of the array.

The DATA statement cannot be used to assign values to elements in blank common or to dummy arguments. Elements in a labeled common block can be initialized with a data initialization statement in any program unit; furthermore, different parts of a block can be initialized in different program units as well as with different statements in the same program unit.

MIXED MODES IN DATA INITIALIZATION STATEMENTS

Mixing of modes between list elements and constants is allowed. The following table shows the legal combinations and the mode of the constant after conversion.

Data \ Constant Element \	Integer	Real	Double Precision	Complex	Logical	Character/ Hollerith	Hexadecimal
Integer	Integer	Integer	Integer	Integer	Illegal	Character	Hexadecimal
Real	Real	Real	Real	Real	Illegal	Character	Hexadecimal
Double Precision	Double Precision	Double Precision	Double Precision	Double Precision	Illegal	Character	Hexadecimal
Complex	Complex	Complex	Complex	Complex	Illegal	Character	Hexadecimal
Logical	Illegal	Illegal	Illegal	Illegal	Logical	Character	Hexadecimal
Character	Illegal	Illegal	Illegal	Illegal	Illegal	Character	Hexadecimal

CHARACTER, HOLLERITH, AND HEXADECIMAL DATA

The initialization rules for character, Hollerith, and hexadecimal constants follow:

CHARACTER OR HOLLERITH CONSTANT

Character variable or character array element:

Requires a character/Hollerith constant whose length must be less than or equal to that of the list item.

Character array of n elements:

Requires n character/Hollerith constants, each must be of a length less than or equal to that of an array element.

Non-character variable or array element:

The character/Hollerith constant must be of a length less than or equal to the number of characters that may be contained in the storage required by the list item.

Non-character array:

Must be last item in list k, and the length of the character constant must not exceed the number of characters that may be contained in the storage required by the array.

The j* specification may not be used in a non-character array.

If the number of characters in the character/Hollerith constant is less than the number of characters defined by the variable list element, the constant will be treated as though an appropriate number of blank characters had been added to the right-hand side of the constant.

If the number of characters in the character/Hollerith constant is greater than the number of characters defined by the variable list element, the constant will be truncated on the right-hand side and a warning error message will be issued.

HEXADECIMAL CONSTANT

If the number of bits in the hexadecimal constant is less than the number of bits defined by the variable list element, the constant will be treated as though an appropriate number of zero bits had been added to the left-hand side of the constant.

If the number of bits in the hexadecimal constant is greater than the number of bits defined by the variable list element, the constant will be truncated on the left-hand side and a warning error message will be given.

An executable program consists of one main program, any number of subprograms, and any number of other external procedures. An executable program is usually a self-contained computing procedure.

A program unit is either a main program or a subprogram consisting of FORTRAN statements and optional comments, terminating with an END line. A program unit containing no FORTRAN statements other than comments and an END line is considered to be a null program; it is diagnosed and executed as if it contained a STOP statement.

MAIN PROGRAM AND SUBPROGRAMS

A FORTRAN program may be written with or without subprograms. One main program is required in any executable FORTRAN program; any number of subprograms may be included.

A main program should begin with the PROGRAM statement. An executable subprogram must begin with either a FUNCTION or SUBROUTINE statement. A specification subprogram must begin with a BLOCK DATA statement.

PROGRAM STATEMENT

Form

PROGRAM progname(p_1,p_2, . . . p_n)

Element Definition

progname Must be a unique symbolic name within the main program. It will be the entry point name and the object module name.

p Each p is a file information parameter required for each input/output file used by the main program and by all subprograms. Each p assumes one of the following forms:

UNITi=filename
TAPEi=filename
INPUT
OUTPUT
PUNCH

i Is a logical unit number in the range 1-99

filename Is a 1-8 character name identifying the file. Maximum number of files is 16.

The form UNITi=filename allows the FORTRAN input/output library module to associate filename with logical unit number i. The form TAPEi=filename serves the same function. The crucial difference is that UNIT identifies a mass storage file and TAPE identifies a tape file.

When a program uses PRINT, PUNCH, or READ statements, the corresponding file names OUTPUT, PUNCH, or INPUT must appear in the PROGRAM statement.

A main program can contain any statement except:

Another PROGRAM statement

BLOCK DATA

FUNCTION

SUBROUTINE

ENTRY

RETURN

Any statement, such as a CALL, that would attempt to reference the program being defined.

A main program must either have a STOP statement or call a subprogram that has a STOP statement.

SUBPROGRAMS

A subprogram is defined by a subprogram header statement: BLOCK DATA, FUNCTION, or SUBROUTINE. The header statement either must be the first statement of a source deck or must immediately follow an END statement of a preceding program unit.

A subprogram headed by a FUNCTION or SUBROUTINE statement is called a procedure subprogram.

Procedure subprograms may be subroutines or functions. Function subprograms return a single value to the expression containing the function's name. The four kinds of functions are:

Statement functions, FUNCTION subprograms } User defined

Intrinsic functions (in-line functions), library functions } System supplied

Subroutine subprograms can return a number of values (or none); they are referenced by a CALL statement. They may be:

User subroutines

Library subroutines

Subprograms are defined separately from the calling program and may be compiled independently of the main program. They are complete program units conforming to all rules of FORTRAN programs. The term program unit refers to either a main program or a subprogram.

A subprogram can call other subprograms; but it cannot call itself directly or indirectly. For example, if program A calls program B, B should not call A. A calling program is a program unit which calls a subprogram.

Subprogram definition statements declare certain names to be the arguments of the subprogram – they are called dummy arguments. They are used as ordinary names within the defining subprogram and indicate the number, type, and order of the arguments and how they are used. The parameters in a subroutine call or a function reference are actual parameters. Actual parameters are expressions which should agree in type with the corresponding dummy arguments in the subprogram definition. The dummy arguments have the value of the actual arguments when the subprogram is executed. Dummy arguments and subprogram name must not appear in COMMON, EQUIVALENCE, or DATA statements.

DEFINING A STATEMENT FUNCTION

Form

$fname(d_1,d_2, \ldots d_n) = expr$

Element Definition

fname	Function name; the function type is determined by the type of this symbolic name
d	Each d is a dummy argument which must be a simple variable
expr	Any expression conforming to the rules for expressions used in assignment statements. It can contain references to library functions, other previously defined statement functions, or function subprograms. Names in the expression which are not dummy arguments have the same value as they would have outside the function when the function is referenced.

The definition of a statement function is contained in a single statement, and it applies only to the program unit which contains the definition.

Statement function names must not appear in DIMENSION, DATA, EQUIVALENCE, COMMON, or EXTERNAL statements. They can appear in a Type statement, but cannot be dimensioned or given an initial value. If the function name is type logical or character, the expression must be the same type. For other types, if the function name and expression differ, conversion is performed as a part of the statement function.

A statement function must precede the first executable statement in the program unit and must follow all specification statements.

REFERENCING STATEMENT FUNCTIONS

A statement function is referenced when the name of the function appears in an expression. An actual argument is any expression of the same type as the corresponding dummy argument.

DEFINING A FUNCTION SUBPROGRAM

Form

typ FUNCTION fname(d_1,d_2, . . . d_n)

Element Definition

typ — Type declaration: INTEGER, REAL, DOUBLE PRECISION, COMPLEX, LOGICAL, CHARACTER†, or omitted.

fname — Function name. If typ is omitted, the function type is determined by the type of the symbolic name fname.

d — Each d is a dummy argument which can be a variable name, array name, or external procedure name. At least one dummy argument is required; no two dummy arguments can have the same name.

The FUNCTION subprogram is a FORTRAN subprogram consisting of any number of statements. Since it is written independently, except for association through the arguments and COMMON, names and statement numbers bear no relationship to names used in other program units. The FUNCTION subprogram is executed whenever its name is referenced in an expression in another program unit.

A function subprogram begins with a FUNCTION statement and ends with an END statement. It returns control to the calling program when a RETURN statement in the function subprogram is executed. The actual arguments in the calling program can be any expression of the same type as the corresponding dummy argument. Effectively, when the function is called the dummy arguments have the values of the actual arguments at the time the function is referenced. Execution of the FUNCTION subprogram returns a single value to the referencing expression through the function name.

The fname must appear as a variable name in the defining function subprogram, and this variable must be given a value at least once in every execution of the subprogram. Within the function, the variable fname can be referenced as a simple variable and redefined. The value returned to the expression referencing the function is the value of fname upon execution of a RETURN statement. If typ is omitted in the FUNCTION statement, fname can occur in a Type statement within the function subprogram. Otherwise fname must not appear in any non-executable statement within the function definition subprogram.

---

†If type is CHARACTER, name can be followed by *n where n is a decimal number specifying the length in bytes returned when the function name is referenced. When *n is not specified the assumed length is one. Whenever the dummy arguments are CHARACTER, they must be declared by a Type statement in the function definition.

Dummy arguments corresponding to actual arguments in the calling reference, which are either array elements or simple variable names, can be given values and redefined within the function subprogram to return results in addition to the value of the function. When dummy arguments are arrays and correspond to array names in the calling references, any elements within the array can be given values or redefined.

When the same variable is used for two or more arguments in a function reference, the corresponding dummy arguments should not be given values within the subprogram definition. Likewise, when an argument in a function reference is in common, the corresponding dummy argument should not be given a value within the subprogram.

REFERENCING EXTERNAL FUNCTIONS

A function or entry point into a function is referenced by using its name together with its argument list, enclosed in parentheses, as a primary in an expression. The actual arguments, which constitute the list, must agree in order, number, type, and length with the corresponding dummy arguments in the defining function subprogram. The one exception is: actual arguments which are character or Hollerith constants also can correspond to dummy arguments of type other than character. Each actual argument in an external function reference must be:

An expression†

An array name

The name of an external procedure

If an actual argument is an external function name or subroutine name, that name must appear in an EXTERNAL statement in the referencing program unit. Furthermore, the corresponding dummy argument must be used as an external function reference or as a subroutine name in a CALL statement.

DEFINING A SUBROUTINE SUBPROGRAM

Form

SUBROUTINE sname(d_1,d_2, . . . d_n)

Element Definition

sname	Symbolic name of the subroutine subprogram
d	Each d is a dummy argument following the same rules as for dummy arguments in the FUNCTION statement, or each d is an * denoting a return point specified by a statement number in the call. Parameters can be omitted entirely for a subroutine, in which case the parentheses must also be omitted.

†Variable names, array element names, and constants are expressions of simple form.

The subroutine subprogram and the function subprogram are independent of the main program. Except for association through the arguments and through common, names and statement numbers used in the subroutine subprogram bear no relationship to names used in other program units. The subroutine subprogram is executed whenever it is referenced in a CALL statement.

A subroutine subprogram begins with a SUBROUTINE statement and ends with an END statement. It returns control to the calling program unit when a RETURN statement is executed.

The sname must not appear in any statement in the subroutine definition except the SUBROUTINE statement.

The rules for dummy arguments (except when * occurs as a dummy argument) are the same as those for function subprograms.

Whenever an * occurs as a dummy argument in the SUBROUTINE statement, in the corresponding position in the CALL statement there must be a statement label as an actual argument. In a CALL statement, an argument is a statement label if it is an & followed by the digits required to comprise a valid label. When an actual argument in a calling program is a FORMAT statement number, the corresponding dummy argument must be an array name in the subroutine.

If the actual argument is a NAMELIST name, the dummy argument must not be an array name; that name should be used only as a NAMELIST name in input/output statements. Furthermore, the elements of that NAMELIST name are elements of the calling program even though the input/output statement occurs in the subprogram.

RETURN STATEMENT

Form

RETURN i

Element Definition

i	Integer constant or variable whose value denotes the nth * in the dummy argument list; i is not allowed in function subprograms

Execution of this statement in a subprogram causes control to return to the calling program unit. In a function, control returns with the function value to the referencing expression. In a subroutine, control returns to the first executable statement following the CALL statement when i is omitted; when i is specified, control returns to the statement label associated with the ith * in the SUBROUTINE statement. If i is out of range, control is returned as though i were not specified.

REFERENCING SUBROUTINE SUBPROGRAMS

Form

CALL sname(p_1,p_2, . . . p_n)

Element Definition

sname — Name of the subroutine being called.

p — Each p is an actual argument of any of the forms described for a function reference. Each p can also take the form &n where n is a statement number. If the SUBROUTINE statement for sname includes no parameters, the parameter list and parentheses are omitted.

Execution of the CALL statement transfers control to the subroutine subprogram or to an entry point in a subroutine subprogram. The actual arguments, which constitute the argument list, must agree in order, number, type, and length with the corresponding dummy arguments in the defining subroutine subprogram. The only exception is that actual arguments which are character or Hollerith constants can also correspond to dummy arguments of type other than character.

Control normally returns to the next executable statement following the CALL statement in the calling program unit. If statement labels are passed as arguments, the subroutine can select alternative returns of the form RETURN i.

Results from the subroutine are returned to the calling program unit (or other program units) when the subroutine changes the values of elements in COMMON, changes values of the arguments, or writes the results to a logical input/output device.

An executing FORTRAN program generally operates on data external to the program itself, so that different sets of data can be manipulated by the same unchanged program. A meaningful FORTRAN program also stores the results it has generated. For input/output, the compiler uses the following information:

Input/Output Unit or Device Ordinal

The operating system associates this number (1-99 decimal) with a particular I/O device (see PROGRAM statement, section 7). Default usually will be a dedicated card reader for input and a dedicated line printer for output.

Format Specification

The format specifies the type of translation required between input data and internal storage and between internal storage and output data (see FORMAT statement, section 9). The format is specified by reference to the statement label (1-99999 decimal) of a FORMAT statement in the program unit containing the I/O command, by reference to the name of an array containing the format specification, or by special reference to input data in the case of NAMELIST. Absence of a format specification results in no conversion. Input data must be in binary form, and output data remain in binary form.

List of Variables

This list contains the names of variable to be input or output. When an array name is included in the list, the entire array is input or output in the order in which the array is stored. A subscripted array name causes the element specified to be input or output. Specific elements of an array also may be input or output through the implied DO specification.

In the absence of the list of variables, on input one record is read, and on output one record is written.

End Condition

If a READ statement is executing when the next sequential record of input data is an end-of-file indicator, the variables become undefined, the end-of-file record becomes the preceding record, execution of the READ is abandoned, and control transfers to the statement specified by the END= option. For input, when end-of-file is encountered and no END= option is specified, control passes to the operating system, which terminates the job and issues an appropriate error message.

Error Condition

If a READ statement is executing when a data transfer error occurs, the variables become undefined, the record in error becomes the preceding record, execution of the READ is abandoned, and control transfers to the statement specified by the ERR= option. On input, if a data transfer error occurs when no ERR= option is specified, control passes to the operating system, which terminates the job and issues an error message.

Sets of input and output data are accessed sequentially. The list of variables determines the task for execution of an input or output command. A READ, WRITE, PRINT, or PUNCH statement processes at least one input or output record and transmits values for each variable in the list. When the list of variables is exhausted, execution is considered complete.

FORTRAN RECORD LENGTH

The length of each record input from cards is 80 characters, and the length of each record output to a card punch device must not exceed 80 characters. The length of each output record to be printed must not exceed 137 characters.

INPUT STATEMENTS

READ FORMATTED

Form

READ (u,fmt,END=m,ERR=n) iolist

Element Definition

u	Integer constant or integer variable specifying the input device
fmt	Label of a FORMAT statement or name of an array containing the format specification
END=m	Optional; transfers control to the statement labeled m when end-of-file is encountered
ERR=n	Optional; transfers control to the statement labeled n when a data transfer error occurs
iolist	List of variables and/or arrays to be read sequentially; iolist can be omitted

Examples

```
C      WHERE DECLARATION WAS  R(20), T(12,15), X(5)
       READ (5,5001) A,B,C,D
   12  READ (IN,2,ERR=47) X,Y,Z
       READ (12,11304) R(1),R(5),(R(J),J=8,15)
       READ (99,100,END=901) T(1,15)
  100  READ (1,10)
```

The READ statement transmits data from the specified device u to storage locations named in iolist, according to the format specified by fmt. More than one record of input data can be transmitted under control of the format specification. Input record length is a maximum of 80 characters.

READ UNFORMATTED

Form

READ (u,END=m,ERR=n) iolist

Element Definition

u	Integer constant or integer variable specifying the input device
END=m	Optional; transfers control to the statement labeled m when end-of-file is encountered
ERR=n	Optional; transfers control to the statement labeled n when a data transfer error occurs
iolist	List of variables and/or arrays to be read sequentially; iolist can be omitted

Examples

```
C      WHERE DECLARATION WAS  R(20), T(12,15), X(5)
       READ (5) A,B,C,D
   12  READ (IN,ERR=47) X,Y,Z
       READ (12) R(1),R(5),(R(J),J=8,15)
  531  READ (99,END=901) T(1,15)
       READ (1)
```

The READ statement transmits one record of input data from the specified device u to the storage locations named in iolist, using no format specification. The input record must be in binary form, and no conversion takes place.

READ WITH IMPLIED DEVICE

Form

READ fmt, iolist

Element Definitions

fmt	Label of a FORMAT specification statement or name of an array containing the format specification
iolist	List of variables and/or arrays to be read sequentially; iolist can be omitted

Examples

```
C     WHERE DECLARATION WAS  R(20), T(12,15), X(5)
      READ 5001,A,B,C,D
      READ 2,X,Y,Z
   27 READ 11304,R(1),R(5),(R(J),J=8,15)
      READ 100,T(1,15)
    8 READ 10
```

The READ statement transmits data from the installation default device, usually a card reader to storage locations named in iolist, according to the format specified by fmt. More than one record of input data can be transmitted under control of the format specification. Input record length is a maximum of 80 characters.

OUTPUT STATEMENTS

WRITE FORMATTED

Form

WRITE (u,fmt) iolist

Element Definitions

u	Integer constant or integer variable specifying the output device
fmt	Label of a FORMAT specification statement or name of an array containing the format specification
iolist	List of variables and/or arrays to be output sequentially; iolist can be omitted

Examples

```
C       WHERE DECLARATION WAS  DOG(3), S(25), U(5,2,40)
        WRITE (6,6001) E,F,G,H
        WRITE (IOUT,3) CAT,DOG,COWS
    2   WRITE (13,11305) S(1),(S(K),K=3,9),S(20)
44112   WRITE (15,16) U(1,1,3)
        WRITE (2,11)
```

The WRITE statement transmits data from storage locations named in iolist to the specified device u, according to the format specified by fmt. More than one record of output data can be transmitted under control of the format specification, and the format also can specify literal data to be transmitted. An output record with carriage control character contains a maximum of 137 characters.

WRITE UNFORMATTED

Form

WRITE (u) iolist

Element Definitions

u	Integer constant or integer variable specifying the output device
iolist	List of variables and/or arrays to be output sequentially; iolist can be omitted

Examples

```
C      WHERE DECLARATION WAS  DOG(3), S(25), U(5,2,40)
    56 WRITE (6) E,F,G,H
       WRITE (6) CAT,DOG,COWS
       WRITE (13) S(1),(S(K),K=3,9),S(20)
   204 WRITE (15) U(1,1,4)
       WRITE (2)
```

The WRITE statement transmits one record of output data from storage locations named in iolist to the specified device u, using no format specification. No conversion takes place, and the output record is transmitted in binary form.

PRINT

Form

PRINT fmt, iolist

Element Definitions

fmt	Label of a FORMAT specification statement or name of an array containing the format specification
iolist	List of variables and/or arrays to be output sequentially; iolist can be omitted

Examples

```
C       WHERE DECLARATION WAS  DOG(3), S(25), U(5,2,40)
        PRINT 6001,E,F,G,H
        PRINT 3,CAT,DOG,COWS
   99   PRINT 11305,S(1),(S(K),K=3,9),S(20)
 1001   PRINT 16,U(4,1,2)
        PRINT 11
```

The PRINT statement transmits data from storage locations named in iolist to a line printer, according to the format specified by fmt. More than one record of output data can be transmitted under control of the format specification, and the format also can specify literal data to be transmitted. An output record with carriage control character contains a maximum of 137 characters.

PUNCH

Form

PUNCH fmt, iolist

Element Definitions

fmt	Label of a FORMAT specification statement or name of an array containing the format specification
iolist	List of variables and/or arrays to be output sequentially; iolist can be omitted

Examples

```
C       WHERE DECLARATION WAS  DOG(3)
        PUNCH 6001,E,F,G,H
        PUNCH 3,CAT,DOG,COWS
44211   PUNCH 11305
```

The PUNCH statement transmits data from storage locations named in iolist to a card punch, according to the format specified by fmt. More than one record of output data can be transmitted, and the format also can specify literal data to be transmitted. Output record length is a maximum of 80 characters.

UNIT POSITIONING

REWIND, BACKSPACE, and ENDFILE statements can be used only for sequential input/output devices. Sequential units contain one or more records grouped as a totally ordered sequential set. The initial position of a unit precedes the first record of information, and the end-of-file indicator follows the last record of information.

REWIND STATEMENT

Form

REWIND u

Element Definition

u	Integer constant or integer variable specifying the sequential unit

Examples

```
      REWIND 5
   70 REWIND IOUT
```

Execution of a REWIND cuases the sequential unit specified by u to be positioned at its initial point.

BACKSPACE STATEMENT

Form

BACKSPACE u

Element Definition

u	Integer constant or integer variable specifying the sequential unit

Examples

```
BACKSPACE 5
BACKSPACE IVER
```

Execution of a BACKSPACE causes the sequential unit specified by u to be positioned at the record preceding the current position. If u is at the initial point, BACKSPACE has no effect.

ENDFILE STATEMENT

Form

ENDFILE u

Element Definition

u	Integer constant or integer variable specifying the sequential unit

Examples

```
ENDFILE 6
ENDFILE NX
```

Execution of this statement causes an end-of-file indicator to be written as the last record on the sequential unit.

NAMELIST

The NAMELIST statement permits a list of variable names or array names to be grouped under an identifying NAMELIST name. In input/output operations, reference to a NAMELIST name reads or writes all variables or arrays associated with the NAMELIST name. Any NAMELIST statement must precede the first executable statement in the program unit and must precede any statement function definitions.

NAMELIST STATEMENT

Form

NAMELIST /nlname/nllist
or
NAMELIST /$nlname_1$/$nllist_1$. . ./$nlname_n$/$nllist_n$

Element Definitions

nlname	NAMELIST name following rules for symbolic names
nllist	List of variables and/or arrays associated with nlname; dummy arguments are not permitted

Examples

```
C       WHERE DECLARATION WAS  GARAGE(4), INK(3,2)
        NAMELIST /FARM/BARN,SHED,GARAGE
        NAMELIST /ART/PEN,INK,PAPER /Y/T,U,V,W /GARF/NURF,SURF,INK
```

Each NAMELIST name declared in a NAMELIST statement must be unique within the program unit. No restriction is placed on the number of variable or array names in nllist, and any variable or array name can be associated with more than one NAMELIST name.

NAMELIST INPUT

The NAMELIST name is used to identify a NAMELIST data block, which can contain one or more records. The NAMELIST data block can be used to set the values of the variables and arrays associated with the NAMELIST name.

NAMELIST DATA BLOCK

Form

b&nlnamebnlexp$_1$,nlexp$_2$, . . .nlexp$_n$&END
or
b&nlnamebnlexp$_1$,nlexp$_2$, . . .nlexp$_n$,&END

Element Definitions

b	The character blank
&nlname	Ampersand followed by the NAMELIST name, which must contain no embedded blanks
nlexp	NAMELIST data expression which must take the form:
	nlitem=nlconst
	nlitem — Variable name or array name associated with the NAMELIST name. Array names can be unsubscripted or have unsigned integer constant subscripts. nlitem must contain no embedded blanks.
	nlconst — Constant which agrees in type with nlitem and specifies the value to which nlitem is set. When nlitem designates an array or a number of array elements, nlconst can specify either a number of constants separated by commas or repeat specifications for constants. The repeat specification consists of an unsigned integer constant indicating the number of repeats, an *, and the constant to be repeated.
&END	Terminator of the NAMELIST data block. &END must contain no embedded blanks and must be complete withinin a single record.

Each nlitem in the NAMELIST data block must be associated with the NAMELIST name nlname by previous declaration. If nlitem is an array, no attempt must be made to store values beyond the length of nlitem. The entire NAMELIST data block can extend over a number of records, but the information for any nlitem or any nlconst must be complete within a single record (each part of a complex constant must be complete within a single record). Character constants are the exception to the rule, as they can be continued on a succeeding record. The NAMELIST data block can be used to specify values for some or all of the variables and/or arrays declared as associated with the NAMELIST name nlname, but execution of a NAMELIST READ always causes input of the entire NAMELIST data block.

NAMELIST READ

Form

READ (u,nlname,END=m,ERR=n)
or
READ nlname

Element Definitions

u	Integer constant or integer variable specifying the input device
nlname	NAMELIST name following rules for symbolic names
END=m	Optional; transfers control to the statement labeled m when end-of-file is encountered
ERR=n	Optional; transfers control to the statement labeled n when a data transfer error occurs

Examples

```
      READ FARM
   10 READ (5,ART)
      READ (10,Y,END=80)
 1458 READ (3,GARF,ERR=900)
```

The NAMELIST READ causes records to be input from the specified device u or from the implied device until the NAMELIST data block identified by &nlname is found. All information in the NAMELIST data block is transmitted to the variables and arrays associated with the NAMELIST name.

NAMELIST OUTPUT

Reference to a NAMELIST name in a NAMELIST WRITE, PRINT, or PUNCH statement causes output of all variables and arrays associated with the NAMELIST name. The entire sequence of records produced by a NAMELIST output statement is suitable to be input by a NAMELIST input statement referencing the same NAMELIST name.

NAMELIST WRITE, PRINT, PUNCH

Form

WRITE (u,nlname)
or
PRINT nlname
or
PUNCH nlname

Element Definitions

u	Integer constant or integer variable specifying the output device
nlname	NAMELIST name following rules for symbolic names

Examples

```
1035 WRITE (6,FARM)
     PRINT Y
   5 PUNCH ART
```

The NAMELIST WRITE, PRINT, or PUNCH causes the NAMELIST data block identified by nlname to be output to the device u, to a line printer, or to a card punch. All variables and arrays associated with the NAMELIST name nlname are output according to the order of the NAMELIST statement declaration, and all array elements are output in the order in which arrays are stored.

ENCODE/DECODE

ENCODE and DECODE statements are similar in action to formatted input/output statements, except records are not transmitted to or from output or input devices. Instead, the records are transferred to or from a buffer. The buffer is either a variable or an array, and it can be considered as a sequence of records. The ENCODE statement, acting like a WRITE statement, transfers records to the buffer. The DECODE statement, acting like a READ statement, transfers records from the buffer.

ENCODE STATEMENT

Form

ENCODE (cl,fmt,bufname) iolist

Element Definitions

cl	Length in characters of each record
fmt	Label of a FORMAT specification statement or name of an array containing the format specification
bufname	Variable or array name identifying the buffer into which encoded records are placed
iolist	List of variables and arrays to be encoded; the list can be omitted

Examples

```
      ENCODE (40,1,ALPHA) A,B,C
   19 ENCODE (64,FORM1,GAMMA) X
```

The ENCODE statement transfers one or more records to the buffer bufname by the action of the format specified by fmt on the variables named in iolist. Each record produced must not be longer than cl characters (if less than cl characters, trailing blanks are inserted). Records written by the ENCODE statement are stored contiguously and in the order in which they were created. The first record written is stored at the start of the buffer identified by bufname. The ENCODE statement must not attempt to store records beyond the length of the buffer.

The variable or array name specified by fmt must not be identical to bufname, and no variable or array name in iolist can be identical to bufname.

DECODE STATEMENT

Form

DECODE (cl,fmt,bufname) iolist

Element Definitions

cl	Length in characters of each record
fmt	Label of a FORMAT specification statement or name of an array containing the format specification
bufname	Variable or array name identifying the buffer from which records are decoded
iolist	List of variables and arrays which receive decoded records; the list can be omitted

Examples

```
1036 DECODE (40,2,ALPHA) D,E,F
     DECODE (64,FORM2,GAMMA) Y
```

The DECODE statement transfers one or more records from bufname to the storage locations named in iolist, according to the format specified by fmt. DECODE must not attempt to use more than cl characters of each record. The DECODE statement reads records sequentially from bufname, and must not attempt to read records beyond the length of the buffer bufname.

On execution of a DECODE statement, no variable or array name in iolist can be identical to bufname.

INPUT/OUTPUT LISTS

The entries in the iolist of an input/output statement are separated by commas. The list is transmitted sequentially from left to right. Specification of a variable name in the list causes that variable to be read or written. Array names appearing in the list can be subscripted or unsubscripted. Specification of an unsubscripted array name causes the entire array to be read or written in order. The order of array element storage follows the rule that each subscript to the right will not increment until the subscript to the left has gone through its range. Specification of a subscripted array name causes the referenced array element to be read or written. Some or all of the elements of an array can be read or written when the implied DO specification is used.

IMPLIED DO SPECIFICATION

The standard implied DO specification is used within the iolist of an input/output statement.

Form

. . . , (aname(isub),isub=i_1,i_2,i_3), . . .
or
. . . , ((aname(isub,jsub),isub=i_1,i_2,i_3),jsub=j_1,j_2,j_3), . . .
or
. . . , (((aname(isub,jsub,ksub),isub=$i_1$$i_2$,$i_3$),jsub=$j_1$,$j_2$,$j_3$), ksub=$k_1$,$k_2$,$k_3$), . . .

Element Definitions

aname	Array name
isub	Integer variable name unsubscripted, whose value is changed by the succeeding DO specification for isub
jsub,ksub	Same as isub, but different integer variable names must be used
i_1,j_1,k_1	Integer constants which specify the initial values of isub, jsub, and ksub
i_2,j_2,k_2	Integer constants which specify the terminal values of isub, jsub, and ksub
i_3,j_3,k_3	Optional; integer constants which specify the value of the increment to be applied to isub, jsub, or ksub in execution of the DO loop. Default is 1.

Examples

```
C       WHERE DECLARATION WAS  Q(8), R(10,20), S(10,12,12)
        ...,(Q(I),I=1,5),...
        ...,((R(I,J),I=3,6,3),J=1,12),...
        ...,(((S(L,M,N),L=1,7),M=2,12,2),N=1,10),...
        ...,(R(3,J),J=1,12),...
        ...,((S(L,M,10),L=1,7),M=2,12,2),...
```

The subscripted array name is separated from the implied DO specification by a comma, and the entire expression is enclosed in parentheses. For arrays with multiple subscripts, the use of more than one implied DO nests the specifications. Each implied DO to the right will not increment until the implied DO to the left has gone through the specified range.

If an unsigned integer constant is used as a subscript of aname, the implied DO specification must not appear.

Unformatted information appears as strings of binary word values in the form in which they normally appear in storage. Formatted information appears as strings of digits, letters, or characters in a form that can be interpreted easily by the user. A FORMAT statement is required to convert input data to internal representations and to convert internal storage values to external output representations.

FORMAT STATEMENT

Form

snumb␣FORMAT(fs_1,fs_2, . . . , fs_n)

Element Definitions

snumb	Required; statement label referenced by an input/output statement
fs	Field specification

Examples

```
21203 FORMAT(I5,I2,F5.2)
   12 FORMAT(21H WEIGHTS AND MEASURES/5X,12G20.5)
4     FORMAT(5A8,F10.4,2I2)
```

The FORMAT statement is non-executable, but it must have a statement label. It can appear anywhere within the program unit in which it is referenced. Field specifications are separated by commas or slashes (which demarcate formatted records); and the field specifications can be grouped by parentheses. Blanks are not significant except in Hollerith specifications.

The FORMAT statement converts input data with no regard for the type of the variable that receives the value. The format field specifications also convert on output with no regard for the type of variable to be output. In general, each format field specification should match the type of each variable of the input/output list.

FIELD DESCRIPTORS

Form

srFw.d	$nHh_1h_2 \ldots h_n$
srEw.d	$'h_1h_2 \ldots h_n'$
srGw.d	nX
srDw.d	Tp
rIw	rZW
rLw	
rRw	
rAw	

Element Definitions

F, E, G, D, I, L, R, A, H, X, T, Z and the apostrophes are called the conversion codes; they indicate the manner of conversion and editing between internal and external representations.

w and n	Non-zero integer constants representing field width in the external character string.
d	Integer constant representing number of digits in fractional part of the external character string (except for G conversion code)
r	Repeat count; optional non-zero integer constant indicating the number of times to repeat the succeeding basic field descriptor
s	Optional; represents a scale factor designator
h	Character that can be represented by the processor
p	Non-zero integer constant indicating character position within record

For all descriptors, except T and the apostrophe descriptor, the field width must be specified, even if it is zero (except for a G descriptor associated with integer, logical, or character type items). Further, w must be greater than or equal to d.

Basic field descriptor is used to signify the field descriptor unmodified by s or r.

External fields are internally represented as constants of the corresponding type.

REPEAT SPECIFICATIONS

Repetition of individual field descriptors (except nH, apostrophe, nX and Tp) is indicated through the repeat count. If the input/output list warrants, conversion will be used the number of times specified by r.

A group is formed by enclosing field descriptors, field separators, or groups within parentheses. The parentheses enclosing the format specification are not considered group delineators.

When a group of field descriptors or field separators is to be repeated, an integer constant group repeat count precedes the left parenthesis to indicate how many times the group is to be interpreted. If not specified, a group repeat count of one is assumed.

FORMAT CONTROL INTERACTION WITH INPUT/OUTPUT LIST

Execution of formatted READ/WRITE statements or ENCODE/DECODE statements initiates format control. Format control depends on information provided jointly by an element of the input/output list and the field descriptor obtained from the format specification.

When a formatted READ statement is executed, one record is read when the format control is initiated; thereafter, additional records are read only as the format specification demands. Such action must not require more characters than a record contains.

When a formatted WRITE statement is executed, an external record is built as the format specification and the input/output list are processed. This external record is output when the format specification demands that a new record be started. Termination of format control causes the current record to be written. A slash in the format specification demands a new record start or the preceding record terminate.

Except for the effects of repeat counts, the format specification is interpreted from left to right.

Each I, F, E, G, D, R, A, L, or Z basic descriptor interpreted in a format specification associates with one element specified by the input/output list; complex elements require interpretation of two F, E, or G basic descriptors. For the H, apostrophe, X, or T descriptors, no corresponding element is specified by the input/output list, and the format control communicates directly with the record.

During a READ operation, any unprocessed characters of the current input record are ignored when format control terminates or when a slash is encountered.

Upon encountering an I, F, E, G, D, A, R, or L descriptor in a format specification, the format control determines if a corresponding element is specified by the input/output list. If so, converted information is transmitted. When the input/output list is exhausted, format control terminates.

When the last outer right parenthesis of the format specification is encountered, a test is made to determine if another list element is specified. If not, control terminates. If another list element is specified, the format control demands a new record start and control returns to the group repeat specification terminated by the last preceding right parenthesis (if none exists, to the first left parenthesis of the format specification). This action has no effect on the scale factor.

SCALE FACTOR

The optional scale factor designator is defined for use with F, E, G, and D conversions (except when G is used on integer, logical, or character) and takes the form nP; n is an optionally signed integer constant.

A scale factor of zero is established when each format statement is first referenced; it holds for all F, E, G, and D field descriptors until another scale factor is encountered. Once a scale factor is specified, it holds for all D, E, F, and G specifications in that FORMAT statement until another scale factor is encountered. To nullify this effect for subsequent D, E, F, and G specifications, a zero scale factor, 0P must precede a specification.

The scale factor n affects conversion as follows:

For F, E, G, and D input conversions with no exponent in the external field, as well as F output conversions, the scale factor sets the externally represented number to an internally represented number times ten raised to the nth power.

For F, E, G, and D input with an exponent in the external field, the scale factor has no effect.

For E and D output, the basic real constant part of the output quantity is multiplied by 10^n and the exponent is reduced by n.

For G output, the effect of the scale factor is suspended unless the magnitude of the data is outside the range that permits effective use of F conversion. If E conversion is required, the scale factor has the same effect as with E output.

NUMERIC CONVERSIONS

The numeric field descriptors I, F, E, and D are used to specify input/output of integer, real, double precision, and complex data. The G descriptor also may be used for numeric conversion.

With all numeric input conversions, leading blanks are not significant and other blanks are zero. Plus signs can be omitted.

With F, E, G, and D input conversions, a decimal point in the input field overrides the decimal point specification supplied by the field descriptor.

With output conversions, the output field is right justified. If the number of characters produced by the conversion is smaller than the field width, leading blanks are inserted in the output field.

With output conversions, the external representation of a negative value is signed.

The number of characters produced by an output conversion must not exceed the field width w

INTEGER CONVERSION

Iw

The numeric field descriptor Iw indicates the external field occupies w positions as an integer. The value of the external field is stored internally as integer.

In the external input field, the character string is an optionally signed integer constant. Embedded blanks are zeros.

REAL CONVERSION

Three conversions are available for use with real data: F, E, and G.

Fw.d

The numeric field descriptor Fw.d indicates the external field occupies w positions, the fractional part of which consists of d digits. The value of the external field is stored internally as real.

The basic form of the external input field consists of an optional sign followed by a string of digits which may contain a decimal point. The basic form can be followed by an exponent in one of the forms:

Signed integer constant

E followed by optionally signed integer constant

D followed by optionally signed integer constant

An exponent preceded by D is equivalent to an exponent preceded by E.

The external output field consists of optional leading blanks, a minus sign if the internal value is negative, and a string of digits, containing a decimal point, which represent the internal value modified by any established scale factor and rounded to d fractional digits.

Ew.d

The numeric field descriptor Ew.d indicates the external field occupies w positions, the fractional part of which consists of d digits. The value of the external field is stored internally as real.

The standard form of the external output field for a scale factor of zero is:

$b.a_1 \ldots a_d E\pm ee$	For values where the magnitude of the exponent is less than 100
$b.a_1 \ldots a_d E\pm eee$	For values where the magnitude of the exponent is 100 to 999
$b.a_1 \ldots a_d E\pm eeee$	For values where the magnitude of the exponent is greater than 999
	b is a minus sign if the number is negative, and blank if the number is positive.
	$a_1 \ldots a_d$ are the d most significant digits of the value correctly rounded.
	Each e is a digit of the decimal exponent.

A scale factor shifts the decimal point so that $a_1 \ldots a_d$ is multiplied by 10^n and the decimal exponent is reduced by n .

If $n \leq 0$, there will be exactly -n leading zeros with d+n significant digits after the decimal point.

If $n > 0$, there will be exactly n significant digits to the left of the decimal point and d-n+1 to the right of the decimal point.

Gw.d

The numeric field descriptor Gw.d indicates the external field occupies w positions with d significant digits. The value of the external field is stored internally as real.

Input processing is the same as for the F conversion.

The method of representation in the external output string is a function of the magnitude of the real data being converted. Let N be the magnitude of the internal data. The following tabulation exhibits a correspondence between N and the equivalent resulting method of conversion:

Magnitude of Data	**Equivalent Conversion**
$0.1 \leq N < 1$	F(w-4).d,4X
$1 \leq N < 10$	F(w-4).(d-1),4X
.	.
.	.
.	.
$10^{d-2} \leq N < 10^{d-1}$	F(w-4).1,4X
$10^{d-1} \leq N < 10^{d}$	F(w-4).0,4X
Otherwise	sEw.d

The effect of the scale factor is suspended unless the magnitude of the data is outside the range that permits effective use of F conversion

DOUBLE PRECISION CONVERSION

Dw.d

The numeric field descriptor Dw.d indicates the external field occupies w positions, the fractional part of which consists of d digits. The value of the external field is stored internally as real. The basic form of the external input field is the same as for real conversions.

The external output field is the same as for E conversion, except the character D, rather than E, precedes the exponent.

COMPLEX CONVERSION

Since complex data consists of a pair of separate real data, the conversion is specified by two real field descriptors interpreted successively – the first for the real part – the second for the imaginary part.

LOGICAL CONVERSION

Lw

The logical field descriptor Lw indicates the external field occupies w positions as a string of information, defined below. The value of the external field is stored internally as logical.

The external input field consists of leading blanks, decimal point, T (for true) or F (for false), and optional trailing characters.

The external output field consists of w-1 blanks followed by T or F.

The G field descriptor also may be used for logical conversion; Gw is the equivalent of Lw. If Gw.d is used the .d is ignored.

CHARACTER CONVERSION

Character information is transmitted through three field descriptors, Aw, Gw, and Rw.

Aw

The Aw descriptor causes w Hollerith characters to be read into, or written from, a specified list element.

Let c1 be the character length of the list element. If the field width specified for A input is greater than or equal to c1, the rightmost c1 characters will be taken from the external input field. If the field width is less than c1, w characters will appear left justified with c1-w trailing blanks in the internal representation.

If the field width specified for A output is greater than c1, the external output field will consist of w-c1 blanks followed by c1 characters from the internal representation. If the field width is less than or equal to c1, the external output field will consist of the leftmost w characters from the internal representation.

Gw

The G descriptor also can be used for transmitting character information only if the corresponding data list element is type character. Gw is the equivalent of Aw. If Gw.d is used, the d is ignored.

Rw

The Rw descriptor causes w Hollerith characters to be read into, or written from a specified list element.

Let cl be the character length of the list element. If the field width specified for R input is greater than or equal to cl, the rightmost cl characters will be taken from the external input field. If the field width is less than cl, w characters will appear right justified with cl-w leading zeros in the internal representation.

If the field width specified for R output is greater than cl, the external output field will consists of w-cl zeros followed by cl characters from the internal representation. If the field width is less than or equal to cl, the external output field will consist of the rightmost w characters from the internal representation.

HOLLERITH FIELD DESCRIPTOR

Hollerith information can be transmitted through four field descriptors, Aw, Rw, nH, and $'h_1h_2 \ldots h_n'$.

nH

The nH descriptor causes Hollerith information to be read into, or written from, the n characters (including blanks) following the nH descriptor in the format specification.

APOSTROPHE FIELD DESCRIPTOR

The apostrophe descriptor causes character information to be read into, or written from the characters (including blanks) between the two apostrophes. If the apostrophe character itself occurs within the apostrophe delimiters, it must be written as two consecutive apostrophes.

Character information may not be read into an apostrophe descriptor containing two consecutive apostrophes.

BLANK FIELD DESCRIPTOR

nX

The field descriptor for blanks is nX.

On input, n characters of the external input record are skipped.

On output, n blanks are inserted in the external output record.

TABULATION DESCRIPTOR

Tp

The Tp descriptor specifies that character position p in the external record is where the next external field begins. Conversion, under format control, continues at character position p until another T descriptor is encountered or until processing begins on the next external record.

p can be either greater than or less than the character position currently being processed, but it must not exceed the record length.

On output, if the same character position is defined more than once, the latest definition will take effect. Because of carriage control, the actual print position is at p-1 when the output is printed.

HEXADECIMAL DESCRIPTOR

Zw

The Zw field descriptor indicates the external field occupies w positions.

On input w hexadecimal digits are transmitted to the associated list element right justified and zero filled. Leading, embedded, and trailing blanks in the input field are treated as zeros. If w is greater than the number of hexadecimal digits that can be represented in the list element, the input string is truncated on the left.

On output, w hexadecimal digits are transmitted from the list element to the output field. If w is less than the number of hexadecimal digits in the list element, the rightmost w digits are output. If w is greater than the number of hexadecimal digits in the list element, the output field is right justified and blank filled.

FORMAT SPECIFICATION BY ARRAY

Any formatted input/output statement can reference an array name instead of a FORMAT statement label. The format specification in the array (beginning with a left parenthesis and ending with a right parenthesis) must constitute a valid format specification. Any information after the right parenthesis ending the format specification is ignored.

The format specification can be inserted in the array by a DATA statement, by a READ statement together with an A format, or by a character assignment statement.

PRINT CARRIAGE CONTROL

The carriage control character does not exist for output records transmitted to a card punch, rotating mass storage, magnetic tape device, or any output device other than the line printer.

The first character of an output record to be printed is used as the carriage control character, which is not printed but controls vertical spacing of the printer. The following values are standard for FORTRAN carriage control for line printers:

Character	Action
blank	Single line feed
0	Double line feed
1	Feed to first line of next page
+	No line feed

Failure to specify a carriage control character may cause unexpected results, because the first good character of output data would be used as the carriage control character.

PROGRAM OPERATION UNDER STAR OPERATING SYSTEM I-10

Control statements direct the STAR Operating System to take specified actions for the user. If the user is communicating interactively with the operating system, control statements are entered individually. If the user runs a job in batch mode, the control statement cards are stored as a file. Execution of the batch processor causes execution of each control card.

All cards in the control card record of a job have the same general format. The first element must be a keyword of one to eight characters. Parameters may follow on the control card; their formats are determined by the keyword. Standard separator characters (between parameters, or between the keyword and parameters) may be any of the following:

(, / = + -

Blanks can precede keywords. Blanks to the right of the last character are ignored.

Control card information always terminates with a period or right parenthesis. If no terminator appears on the first control card, the system assumes control information is continued on the next card, starting in column one.

For the proper control card setup for a batch job, see the STAR Operating System Reference Manual (publication no. 60384400).

FORTRAN CONTROL CARD

The STAR FORTRAN control card is of the form:

FORTRAN (I=fn1,B=fn2,L=fn3,OPT=optlist)

fn1	Name of a physical file containing the FORTRAN program to be compiled
fn2	Name of a physical file to hold the compiler generated object decks or modules
fn3	Name of a physical file to hold the compiler generated listings and output
optlist	Any combination, in any order of the letters A, B, C, L, M, O, V, and Y.

All three files are of type physical and must have been created before control is transferred to the compiler. These files may be created by the CREATE utility program provided with STAR-OS. This utility provides a simple way to create a file with a control statement.

The compiler then opens these files at hexadecimal addresses 10000000, 30000000, and 50000000 for fn1, fn2, and fn3, respectively.

COMPILE OPTIONS

All compiler options can be used in any combination and in any order:

Option	A	Requests the assembly listing
	B	Requests the compiler to build the object file
	C	Requests cross reference listing of all labels and symbolic names
	L	Suppresses the source listing
	M	Requests a memory map of all storage and register assignments for variables and arrays
	O	Requests the compiler to optimize object code. This option causes the optimization of object code at the expense of a longer compile time
	V	Requests implicit vectorization of all DO loops satisfying the conditions for vectorization
	Y	Requests a fast syntax check. The source listing and all error diagnostics of a syntactical nature are produced.

EXAMPLE OF A FORTRAN COMPILATION AT THE TERMINAL

```
LOGON 999997 A 400SDS
CREATE(PRINT,30,U=1,T=P) / 100 I
CREATE(OBJ,50,U=2,T=P) / 45 I
-----------------------------------------------------------
         USER READS IN A CARD DECK TO FILE FTNTST00
-----------------------------------------------------------
FORTRAN(I=FTNTST00,L=PRINT,B=OBJ,OPT=ABCM) / 1000 I
GIVE(PRINT,U=999999) / 20 I
LOGOFF
```

SAMPLE PROGRAMS II-1

The following sample programs are coordinated with the reference section to provide specific examples of FORTRAN statements as they can appear in complete programs.

The sample programs were run using the compiler options A, B, M, O, and V. Only the pages of output showing the source listing and the generated results are reproduced.

PROGRAM HIERARC

This program illustrates expression evaluation as described in part I, section 3. The program does the following:

One NAMELIST name, called DDD, is declared to represent the four integer variables I1, I2, I3, and I4, and the six real variables R1, R2, R3, R4, R5, and R6.

Each variable is set to the value of an expression. Each expression is evaluated.

The NAMELIST WRITE statement prints the values of all variables in the NAMELIST list DDD, and the program terminates.

The rules for evaluation of expressions and the rules for assignment of variables are both involved in the following explanations.

The value of I1 is 4 after evaluation of the expression and assignment. The division was evaluated first, and the result was 0 because the result 0.666 . . . was truncated. The result of four plus zero was four.

The value of I2 is identical to I1 after evaluation and assignment. The division was performed first, with 0.666 . . . as the result. The result of 4 plus 0.666 . . . was 4.666 . . . , but the final result was assigned to an integer variable and truncated.

The value of I3 after evaluation and assignment was 65536 . Note that within the group, evaluation is from right to left, therefore the result represents the number 4 to the eighth power.

The value of I4 after evaluation and assignment is 37 . The exponentiation is evaluated first, with the result 49 . Note that the minus sign is an operator on the number 5 and the following term 7**2 , not a property of the constant seven.

For R1 and R2, the evaluations are identical to the evaluations of I1 and I2, and the assignments float the results.

The value of R3 after evaluation and assignment is a real number very close to the value 43046721 . Evaluation of the exponentiations was right to left. The result would have been exactly equal to three to the eighth power, but the calculations were done in real mode because a real constant was in the first term evaluated.

The value of R4 is 57.35 after evaluation and assignment. The divisions are done left to right, yielding results of 2.7 and 1.35 . The result of the addition is assigned.

The value of R5 is 2.5 after evaluation and assignment. Multiplication and division are in the same hierarchical group and are handled from left to right. One times three is three, divided by two is one, times five is five, divided by the real constant 2. is 2.5 .

The value of R6 after evaluation and assignment is 44.0 . The calculations are done in real mode because of the real constant 4.0 .

```
STAR_FORTRAN VER.1.*                    - SOURCE LISTING            11:49 A.M. FRIDAY 26TH. APRIL, 1974.
   00001          PROGRAM HIERARC(OUTPUT,TAPE6=OUTPUT)                                    0001/00001
   00002          NAMELIST/DDD/I1,I2,I3,I4,R1,R2,R3,R4,R5,R6                              0001/00002
   00003          I1=4+2/3                                                                0001/00003
   00004          I2=4+2./3.                                                              0001/00004
   00005          I3=4**2**3                                                              0001/00005
   00006          I4=5-7**2+81                                                            0001/00006
   00007          R1=4+2/3                                                                0001/00007
   00008          R2=4+2./3.                                                              0001/00008
   00009          R3=3**2**4.                                                             0001/00009
   00010          R4=56+5.4/2/2                                                           0001/00010
   00011          R5=1*3/2*5/2.                                                           0001/00011
   00012          R6=4.*7/2+30                                                            0001/00012
   00013          WRITE(6,DDD)                                                            0001/00013
   00014          STOP                                                                    0001/00014
   00015          END                                                                     0001/00015
NUMBER OF LOOPS IN THE PROG =  0000
NO OF UNCOLLAPSABLE  LOOPS  =  0000
```

```
&DDD
I1      =  4,
I2      =  4,
I3      =  65536,
I4      =  37,
R1      =  .4E+01,
R2      =  .4666666666667E+01,
R3      =  .4304672100002E+08,
R4      =  .5735E+02,
R5      =  .25E+01,
R6      =  .44E+02,
&END
**  STOP  **
```

PROGRAM ASSIGN

This program illustrates the assignment statement as described in part I, section 4. The program does the following:

All variable names beginning with the letter D are declared double precision. All variable names beginning with the letter C are declared complex. The character and logical variables are declared explicitly.

Three NAMELIST lists are declared. The first, called INIT, contains the initial values of I (for integer), R (for real), DP (for double precision), and CP (for complex). The second, called SET, holds the variables which represent one variable type set to another. Here the variable names indicate the nature of the variable:

ITOI	means	integer set to integer
CPTODP	means	complex set to double precision
DPTOI	means	double precision to integer

The third NAMELIST list, called SETCL, contains the character and logical variables. The variable name ACH4 indicates a variable of type character, 4 bytes long. The variable name ACH7TO4 indicates a 7-byte character variable set to a 4-byte character variable.

The NAMELIST WRITE statements output the three NAMELIST lists, and the program terminates.

Note that the constant to which DP was assigned should have been followed by a D specification, in this case D+0. Because the D+0 did not follow, the constant was assumed to be single precision, and the last seven digits of precision were lost. Double precision notation for constants and double precision typing for variables must be maintained if the precision is critical. Compare the original value of DP with the value of DPTODP in the output of the NAMELIST list SET.

```
STAR_FORTRAN VER.1.*                    - SOURCE LISTING            11:24 A.M. FRIDAY 26TH. APRIL, 1974.
   00001          PROGRAM VTEST(OUTPUT,TAPE4=OUTPUT)                                          0001/00001
   00002          REAL A(3,3), B(3)                                                           0001/00002
   00003          NAMELIST/AB/A,B                                                             0001/00003
   00004          DO 5 K=1,3                                                                  0001/00004
   00005          B(K)= 3.1                                                                   0001/00005
   00006          DO 5 L=1,3                                                                  0001/00006
   00007    5     A(L,K)= 1.0                                                                 0001/00007
   00008          WRITE(4,AB)                                                                 0001/00008
   00009          A(*,1)=B(*)                                                                 0001/00009
   00010          WRITE(4,AB)                                                                 0001/00010
   00011          A(1:3:1,3)=B                                                                0001/00011
   00012          WRITE(4,AB)                                                                 0001/00012
   00013          A(2,1:3)=B+4                                                                0001/00013
   00014          WRITE(4,AB)                                                                 0001/00014
   00015          STOP                                                                        0001/00015
   00016          END                                                                         0001/00016
 STATEMENT  ON  LINE  NUMBER   0013  HAS RENDERED  FOLLOWING LOOPS UNCOLLAPSABLE
LOOP LINE NO   0013
NUMBER OF LOOPS IN THE PROG =  0005
NO OF UNCOLLAPSABLE  LOOPS  =  0001
```

```
&AB
A       =  .1E+01,  .1E+01,  .1E+01,  .1E+01,  .1E+01,  .1E+01,  .1E+01,  .1E+01,  .1E+01,
B       =  .31E+01,  .31E+01,  .31E+01,
&END


&AB
A       =  .31E+01,  .31E+01,  .31E+01,  .1E+01,  .1E+01,  .1E+01,  .1E+01,  .1E+01,  .1E+01,
B       =  .31E+01,  .31E+01,  .31E+01,
&END


&AB
A       =  .31E+01,  .31E+01,  .31E+01,  .1E+01,  .1E+01,  .1E+01,  .31E+01,  .31E+01,  .31E+01,
B       =  .31E+01,  .31E+01,  .31E+01,
&END


&AB
A       =  .31E+01,  .71E+01,  .31E+01,  .1E+01,  .71E+01,  .1E+01,  .31E+01,  .71E+01,  .31E+01,
B       =  .31E+01,  .31E+01,  .31E+01,
&END
**  STOP  **
```

PROGRAM ARRAY

This program illustrates the order of array storage as described in part I, section 2. The program does the following:

The array A is declared and dimensioned as 50 by 50.

Two DO loops are set up and nested. All elements of the array A are referenced and set to 1.5 , but the elements are referenced in an order which does not reflect internal storage of array A.

Two more DO loops are set up and nested. All elements of A are referenced and set to 1.5 , but the compiler can implement vector optimization because the elements are referenced in the order in which they are stored.

The program produces no output. The program terminates.

Significant improvements in execution time result when arrays are referenced in the order of their internal storage.

```
STAR_FORTRAN VER.1.*                    - SOURCE LISTING              2:02 P.M. FRIDAY 26TH. APRIL, 1974.
   00001          PROGRAM ARRAY(OUTPUT,TAPE6=OUTPUT)                                        0001/00001
   00002          REAL A(50,50)                                                             0001/00002
   00003          DO 10 I=1,50                                                              0001/00003
   00004          DO 10 J=1,50                                                              0001/00004
   00005   10     A(I,J)=1.5                                                                0001/00005
   00006          DO 20 I=1,50                                                              0001/00006
   00007          DO 20 J=1,50                                                              0001/00007
   00008   20     A(J,I)=1.5                                                                0001/00008
   00009          STOP                                                                      0001/00009
   00010          END                                                                       0001/00010
 STATEMENT  ON  LINE  NUMBER   0005  HAS RENDERED  FOLLOWING LOOPS UNCOLLAPSABLE
LOOP LINE NO   0004
LOOP LINE NO   0003
NUMBER OF LOOPS IN THE PROG =  0004
NO OF UNCOLLAPSABLE  LOOPS  =  0002
```

NO OUTPUT

PROGRAM VTEST

This program illustrates the assignment of arrays as described in part I, section 4. The program does the following:

Array A is declared and dimensioned as three by three.

Array B is declared and dimensioned for three elements.

The NAMELIST list AB is declared to include array A and array B.

A nest of loops assigns the value 3.1 to all elements of B and the value 1.0 to all elements of A. The NAMELIST list AB is printed.

The first three elements of A are set to the values of the three elements of B. The NAMELIST list AB is printed.

The last three elements of A are set to the values of the three elements of B. The NAMELIST list AB is printed.

The second, fifth, and seventh elements of A are set to the values of the three elements of B added to the constant 4 . The NAMELIST list AB is printed. The program terminates.

```
STAR_FORTRAN VER.1.*                    - SOURCE LISTING              9:48 A.M. FRIDAY 26TH. APRIL, 1974.
   00001          PROGRAM ASSIGN(OUTPUT,TAPE6=OUTPUT)                                          0001/00001
   00002          IMPLICIT DOUBLE PRECISION(D), COMPLEX(C)                                     0001/00002
   00003          CHARACTER*4 ACH4,ACH4TO7                                                     0001/00003
   00004          CHARACTER*7 ACH7,ACH7TO4                                                     0001/00004
   00005          LOGICAL LOG,ILLOG                                                            0001/00005
   00006          NAMELIST/INIT/I,R,DP,CP                                                      0001/00006
   00007          NAMELIST/SET/ITOI,ITOR,ITODP,ITOCP,RTOI,RTOR,RTODP,RTOCP,                    0001/00007
                 X  DPTOI,DPTOR,DPTODP,DPTOCP,CPTOI,CPTOR,CPTODP,CPTOCP                       0001/00008
   00008          NAMELIST/SETCL/ACH4,ACH4TO7,ACH7,ACH7TO4,LOG,ILLOG                           0001/00009
   00009          I= 47                                                                        0001/00010
   00010          R= 4.361                                                                     0001/00011
   00011          DP= 7.11223344556677889911                                                   0001/00012
   00012          CP= (8.023,.45)                                                              0001/00013
   00013          ITOI=I                                                                       0001/00014
   00014          ITOR=R                                                                       0001/00015
   00015          ITODP=DP                                                                     0001/00016
   00016          ITOCP=CP                                                                     0001/00017
   00017          RTOI=I                                                                       0001/00018
   00018          RTOR=R                                                                       0001/00019
   00019          RTODP=DP                                                                     0001/00020
   00020          RTOCP=CP                                                                     0001/00021
   00021          DPTOI=I                                                                      0001/00022
   00022          DPTOR=R                                                                      0001/00023
   00023          DPTODP=DP                                                                    0001/00024
   00024          DPTOCP=CP                                                                    0001/00025
   00025          CPTOI=I                                                                      0001/00026
   00026          CPTOR=R                                                                      0001/00027

   00027          CPTODP=DP                                                                    0001/00028
   00028          CPTOCP=CP                                                                    0001/00029
   00029          WRITE(6,INIT)                                                                0001/00030
   00030          WRITE(6,SET)                                                                 0001/00031
   00031          ACH4='STAR'                                                                  0001/00032
   00032          ACH7=7HFORTRAN                                                               0001/00033
   00033          ACH4TO7=ACH7                                                                 0001/00034
   00034          ACH7TO4=ACH4                                                                 0001/00035
   00035          LOG=.TRUE.                                                                   0001/00036
   00036          ILLOG=.NOT.LOG                                                               0001/00037
   00037          WRITE(6,SETCL)                                                               0001/00038
   00038          STOP                                                                         0001/00039
   00039          END                                                                          0001/00040
NUMBER OF LOOPS IN THE PROG =  0000
NO OF UNCOLLAPSABLE  LOOPS  =  0000
```

```
&INIT
I       =  47,
R       =  .4361E+01,
DP      =  .71122334455668610377848987D+01,
CP      =( .8023E+01,  .45E+00),
&END


&SET
ITOI    =  47,
ITOR    =  4,
ITODP   =  7,

ITOCP   =  8,
RTOI    =  .47E+02,
RTOR    =  .4361E+01,
RTODP   =  .7112233445567E+01,
RTOCP   =  .8023E+01,
DPTOI   =  .47D+02,
DPTOR   =  .43609999999999899955582805D+01,
DPTODP  =  .71122334455668010377848987D+01,
DPTOCP  =  .80230000000000245563569479D+01,
CPTOI   =( .47E+02,  .0),
CPTOR   =( .4361E+01,  .0),
CPTODP  =( .0,  .0),
CPTOCP  =( .8023E+01,  .45E+00),
&END


&SETCL
ACH4    =  'STAR',
ACH4TO7 =  'FORT',
ACH7    =  'FORTRAN',
ACH7TO4 =  'STAR',
LOG     =  .TRUE.,
ILLOG   = .FALSE.,
&END
**  STOP  **
```

PROGRAM CRANK

This program illustrates the use of the FORMAT statement as described in part I, section 9. The program does the following:

Double precision, character, and logical variable names are declared explicitly.

A double precision constant, which is π to 30 places, is assigned to the variable DPMANGLE. The variable PIMANGLE is a single precision real representation of DPMANGLE. A 5-digit integer is assigned. The next six assignments set variables to integer, real, and double precision representations of zero and one. A character and a logical variable are assigned.

The WRITE statement outputs the variable PIMANGLE in various field widths under control of the F, E, and G field descriptors. The variable DPMANGLE is printed in various field widths under control of the D field descriptor. Note the asterisk which appears when the field width is not great enough.

The WRITE statement outputs the variable IOOOCH in three widths under control of the I field descriptor. The asterisk indicates insufficient field width.

The WRITE statement outputs the internal representations of the integer, real, and double precision constants zero and one under control of the Z field descriptor.

The WRITE statement outputs the variable LEEDGE in L format. Note that the L descriptor generates a representation which is not in the form of a logical constant. The variable CHEERZ is printed five times under control of the A and R field descriptors.

```
STAR_FORTRAN VER.1.*                    - SOURCE LISTING              3:59 P.M. FRIDAY 26TH. APRIL, 1974.
   00001          PROGRAM CRANK(OUTPUT,TAPE6=OUTPUT)                                          0001/00001
   00002          DOUBLE PRECISION DPMANGLE,DPZERO,DPONE                                      0001/00002
   00003          CHARACTER*5 CHEERZ                                                          0001/00003
   00004          LOGICAL LEEDGE                                                              0001/00004
   00005          DPMANGLE= 3.14159265358979323846264338327 9D+0                              0001/00005
   00006          PIMANGLE=DPMANGLE                                                           0001/00006
   00007          IOOOCH= 23754                                                               0001/00007
   00008          IZERO= 0                                                                    0001/00008
   00009          IONE= 1                                                                     0001/00009
   00010          RZERO= 0.                                                                   0001/00010
   00011          RONE= 1.                                                                    0001/00011
   00012          DPZERO= 0.                                                                  0001/00012
   00013          DPONE= 1.                                                                   0001/00013
   00014          CHEERZ=5HURGLE                                                              0001/00014
   00015          LEEDGE=.FALSE.                                                              0001/00015
   00016          WRITE(6,110) (PIMANGLE,I=1,9),(DPMANGLE,K=1,4)                              0001/00016
   00017   110    FORMAT(1H1,10X,'PIMANGLE =',F8.4/20X,'=',F12.6/                             0001/00017
                 X  20X,'=',F20.10//20X,'=',E8.4/20X,'=',E12.6/                              0001/00018
                 X  20X,'=',E20.10//20X,'=',G8.4/20X,'=',G12.6/                              0001/00019
                 X  20X,'=',G20.10//                                                         0001/00020
                 X  10X,'DPMANGLE =',D8.4/20X,'=',D12.6/20X,'=',D20.10/                      0001/00021
                 X  20X,'=',D36.30)                                                          0001/00022
   00018          WRITE(6,130) IOOOCH,IOOOCH,IOOOCH                                           0001/00023
   00019   130    FORMAT(/5X,'IOOOCH =',I3/13X,I5/13X,I10)                                    0001/00024
   00020          WRITE(6,140) IZERO,IONE,RZERO,RONE,DPZERO,DPONE                             0001/00025
   00021   140    FORMAT(/15X,'HEX VALUES =',Z32/ 11(27X,Z32/))                               0001/00026
   00022          WRITE(6,160) LEEDGE,LEEDGE,(CHEERZ,N=1,5)                                   0001/00027

   00023   160    FORMAT(/1X,'LEEDGE =',L1,5X,'=',L7//                                        0001/00028
                 X  30X,'CHEERZ =',A8,'=',A5,'=',A3,'=',R5,'=',R8)                           0001/00029
   00024          STOP                                                                        0001/00030
   00025          END                                                                         0001/00031
 STATEMENT  ON  LINE  NUMBER   0022  HAS RENDERED  FOLLOWING LOOPS UNCOLLAPSABLE
LOOP LINE NO   0022
 STATEMENT  ON  LINE  NUMBER   0016  HAS RENDERED  FOLLOWING LOOPS UNCOLLAPSABLE
LOOP LINE NO   0016
 STATEMENT  ON  LINE  NUMBER   0016  HAS RENDERED  FOLLOWING LOOPS UNCOLLAPSABLE
LOOP LINE NO   0016
NUMBER OF LOOPS IN THE PROG =  0003
NO OF UNCOLLAPSABLE  LOOPS  =  0003
```

```
         PIMANGLE =  3.1416

                  =    3.141593
                  =       3.1415926536

                  =*3142E+0
                  = .3141 59E+01
                  =     .3141592654E+01

                  =*142
                  = 3.14159
                  =     3.141592654

        DPMANGLE =*3142D+0
                  = .3141 59D+01
                  =     .3141592654D+01
                  = .3141592653589793238462643382 29D+01

    IOOOCH =*54
            23754
                 23754

             HEX VALUES =                  0000000000000000
                                           0000000000000001
                                           8000000000000000
                                           FFD2400000000000
                           8000000000000000800000000000000
                           FFD2400000000000800000000000000

LEEDGE =F     =      F

                           CHEERZ =   URGLE=URGLE=URG=URGLE=OOOURGLE
**  STOP  **
```

CHARACTER SET A

The following is the CDC standard ASCII 64-character subset:

Character	Punch	Hexadecimal
b space	no punch	20
! exclamation sign	12-8-7	21
" quote	8-7	22
# number sign	8-3	23
$ dollar sign	11-8-3	24
% percent sign	0-8-4	25
& ampersand	12	26
' apostrophe	8-5	27
(left parenthesis	12-8-5	28
) right parenthesis	11-8-5	29
* asterisk	11-8-4	2A
+ plus sign	12-8-6	2B
, comma	0-8-3	2C
- minus sign	11	2D
. period	12-8-3	2E
/ slash	0-1	2F
0	0	30
1	1	31
2	2	32
3	3	33
4	4	34
5	5	35
6	6	36
7	7	37
8	8	38
9	9	39
: colon	8-2	3A
; semi-colon	11-8-6	3B
< less than sign	12-8-4	3C
= equals sign	8-6	3D
> greater than sign	0-8-6	3E
@ commercial at	8-4	40

Character	Punch	Hexadecimal
A	12-1	41
B	12-2	42
C	12-3	43
D	12-4	44
E	12-5	45
F	12-6	46
G	12-7	47
H	12-8	48
I	12-9	49
J	11-1	4A
K	11-2	4B
L	11-3	4C
M	11-4	4D
N	11-5	4E
O	11-6	4F
P	11-7	50
Q	11-8	51
R	11-9	52
S	0-2	53
T	0-3	54
U	0-4	55
V	0-5	56
W	0-6	57
X	0-7	58
Y	0-8	59
Z	0-9	5A
[left bracket	12-8-2	5B
\ reverse slash	0-8-2	5C
] right bracket	11-8-2	5D
^ circumflex	11-8-7	5E
_ underline	0-8-5	5F

Since collating is done by hexadecimal value, this list represents the collating sequence.

LIBRARY FUNCTIONS B

INTRINSIC FUNCTIONS

Intrinsic Function	Mathematical Definition	Number of Arguments	Symbolic Name	Type of: Argument	Type of: Function
Absolute Value	$\|a\|$	1	ABS	Real	Real
			IABS	Integer	Integer
			DABS	Double	Double
Truncation	Sign of a times largest integer $\leq \|a\|$	1	AINT	Real	Real
			INT	Real	Integer
			IDINT	Double	Integer
Remaindering	$a_1 (\text{mod } a_2)$	2	AMOD†	Real	Real
			MOD†	Integer	Integer
Choosing Largest Value	$\text{Max}(a_1, a_2, \ldots)$	n	AMAX0	Integer	Real
			AMAX1	Real	Real
			MAX0	Integer	Integer
			MAX1	Real	Integer
			DMAX1	Double	Double
Choosing Smallest Value	$\text{Min}(a_1, a_2, \ldots)$	n	AMIN0	Integer	Real
			AMIN1	Real	Real
			MIN0	Integer	Integer
			MIN1	Real	Integer
			DMIN1	Double	Double
Float	Conversion from integer to real	1	FLOAT	Integer	Real
Fix	Conversion from real to integer	1	IFIX	Real	Integer
Transfer of Sign	Sign of a_2 times $\|a_1\|$	2	SIGN	Real	Real
			ISIGN	Real	Real
			DSIGN	Double	Double

†The function MOD or AMOD (a_1, a_2) is defined as $a_1 - |a_1/a_2| a_2$, where $|x|$ is the integer whose magnitude does not exceed the magnitude of x and whose sign is the same as x.

Intrinsic Functions	Mathematical Definition	Number of Arguments	Symbolic Name	Type of: Argument	Function
Positive Difference	a_1-Min(a_1,a_2)	2	DIM IDIM	Real Integer	Real Integer
Obtain Most Significant Part of Double Precision Argument		1	SNGL	Double	Real
Obtain Real Part of Complex Argument		1	REAL	Complex	Real
Obtain Imaginary Part of Complex Argument		1	AIMAG	Complex	Real
Express Single Precision Argument in Double Precision Form		1	DBLE	Real	Double
Express Two Real Arguments in Complex Form	$a_1+a_2\sqrt{-1}$	2	COMPLX	Real	Complex
Obtain Conjugate of a Complex Argument		1	CONJG	Complex	Complex

BASIC EXTERNAL FUNCTIONS

Basic External Function	Definition	Number of Arguments	Symbolic Name	Type of: Argument	Type of: Function
Exponential	e^a	1	EXP	Real	Real
		1	DEXP	Double	Double
		1	CEXP	Complex	Complex
Natural Logarithm	$\log_e(a)$	1	ALOG	Real	Real
		1	DLOG	Double	Double
		1	CLOG	Complex	Complex
Common Logarithm	$\log_{10}(a)$	1	ALOG10	Real	Real
			DLOG10	Double	Double
Trigonometric Sine	sin(a)	1	SIN	Real	Real
		1	DSIN	Double	Double
		1	CSIN	Complex	Complex
Trigonometric Cosine	cos(a)	1	COS	Real	Real
		1	DCOS	Double	Double
		1	CCOS	Complex	Complex
Hyperbolic Tangent	tanh(a)	1	TANH	Real	Real
Square Root	$(a)^{1/2}$	1	SQRT	Real	Real
		1	DSQRT	Double	Double
		1	CSQRT	Complex	Complex
Arctangent	arctan(a)	1	ATAN	Real	Real
		1	DATAN	Double	Double
	$\arctan(a_1/a_2)$	2	ATAN2	Real	Real
		2	DATAN2	Double	Double
Remaindering	$a_1 (\mathrm{mod}\ a_2)$	2	DMOD†	Double	Double
Modulus	$(a^2+b^2)^{1/2}$ for a+bi	1	CABS	Complex	Real
Basic External Function	tan(a)	1	TAN	Real	Real
			DTAN	Double	Double
Arcsine	arcsin(A)	1	ASIN	Real	Real
Arccosine	arccos(A)	1	ACOS	Real	Real

†The function DMOD (a_1,a_2) is defined as $a_1-(a_1/a_2)a_2$, where $|x|$ is the integer whose magnitude does not exceed the magnitude of x and whose sign is the same as the sign of x.

MATHEMATICAL LIBRARY FUNCTIONS DESCRIPTIONS C

Specifications in this appendix are intended as guidelines and are subject to change. The routines included are listed below with the number of the page where each description begins.

ALOG

Purpose: To compute the natural logarithm of a real number.

Usage: Y = ALOG(X)

Where X is the single precision floating point argument, and Y is the result in single precision floating point.

Normal Return: Return with result in Y

Error Messages:
INDEFINITE ARGUMENT IN ALOG
ZERO ARGUMENT IN ALOG
NEGATIVE ARGUMENT IN ALOG

The message is written on the standard output file and displayed on the user's terminal; the result is set to indefinite and a normal exit is taken from ALOG.

Storage: 70 words (shared with ALOG10)

Accuracy: 45 bits approximately

Mathematical Method: $X = 2^n * W$ where $1/2 \leqslant W < 1$

and n is an integer

For X outside the range $(\sqrt{2}/2 \leqslant X < \sqrt{2})$

Let $T = (W - \sqrt{2}/2)/(W + \sqrt{2}/2)$

$LOG_e(X) = (N - 1/2)*LOG_e(2) + LOG_e((1 + T)/(1 - T))$

For X in the range $(\sqrt{2}/2 \leqslant X < \sqrt{2})$

Let $T = (X - 1)/(X + 1)$

$LOG_e(X) = LOG_e((1 + T)/(1 - T))$

Where

$$LOG_e((1 + T)/(1 - T)) \; 1 \; 2T * \sum_{n=0}^{6} CnT^{2n}$$

C0=1.0000 00000 00000 01720 16224 E+00

C1=3.3333 33333 32761 81768 85283 E-01

C2=2.0000 00003 09807 78908 99307 E-01

C3=1.4285 70799 46082 73472 61398 E-01

C4=1.1111 71831 83715 43428 06719 E-01

C5=9.0609 35658 17935 37172 14254 E-02

C6=8.4191 86575 86305 31375 34817 E-02

Reference: "A Study of Mathematical Approximation"
CDC Publication Number 60114500, Rev. A., p. 26

ALOG10

Purpose: To compute the logarithm to the base 10 of a real number.

Usage: Y = ALOG10(X)

Where X is the single precision floating point argument, and Y is the result in single precision floating point

Normal Return: Return with result in Y.

Error Messages: INDEFINITE ARGUMENT IN ALOG10
ZERO ARGUMENT IN ALOG10
NEGATIVE ARGUMENT IN ALOG10

The message is written on the standard output file and displayed on the user's terminal; the result is set to indefinite and a normal exit is taken from ALOG10.

Storage: 70 words (shared with ALOG)

Accuracy: 45 bits approximately

Mathematical Method: $LOG_{10}(X) = LOG_{10}(e)*LOG_e(X)$

Where

$LOG_e(X)$ is computed as described for the function ALOG.

ASINCOS

Purpose: To compute the arcsin or the arccos of a real number.

Usage: Y = ASIN(X) or Y = ACOS(X)

Where X is the single precision floating point argument, and Y is the result in single precision floating point.

Normal Return: Return with result in Y.

Error Messages: INDEFINITE ARGUMENT IN ASIN (ACOS)
ARGUMENT .GT. ONE IN ASIN (ACOS)

The message is written on the standard output file and displayed on the user's terminal; the result is set to indefinite and a normal exit is taken from ASIN.

Storage: 65 words

Accuracy: 45 bits approximately

Mathematical Methods: Y = ASIN(X) X = 0

Y = - ASIN(X) X > 0

ACOS(X) = PI/2 - ASIN(X)

if 0.LE.X.LE.1/2 U = X ASIN(X) = ASIN(U)

if 1/2.LT.X.LE.1 U = SQRT(1 - X/2) ASIN(X) = PI/2 - 2*ASIN(U)

ASIN(U) is calculated from a polynomial of degree 22

References: 6400 FORTRAN Extended Library

Purpose: To compute the arctangent of a real number.

Usage: Y = ATAN(X)

Where X is the single precision floating point argument, and Y is the result in single precision floating point.

Normal Return: Return with result in Y.

Error Messages: INDEFINITE ARGUMENT IN ATAN

The message is written on the standard output file and displayed on the user's terminal; the result is set to indefinite and a normal exit is taken from ATAN.

Storage: 79 words

Accuracy: 45 bits approximately

Mathematical Method: Let A = ATAN(X), then $-PI/2 \leqslant A \leqslant +PI/2$

Let P = TAN(PI/16), T = TAN(3PI/16)

ATAN(X) = sign(X)*ATAN(V), V = ABS(X)

ATAN(V) = ATAN(R) + C, R, C defined below

$0 \leqslant V < P$, R = V, C = 0.0

$P \leqslant V < \sqrt{2}-1$, R = (V - P)/(1 + V*P), C = PI/16

$\sqrt{2} - 1 < V < 1$, R = (V - T)/(1 + V*T), C = 3PI/16

$1 \leqslant V < \sqrt{2} + 1$, R = (V*T - 1)/(V + T), C = 5PI/16

$\sqrt{2} + 1 \leqslant V$, R = (V*P - 1)/(V + P), C = 7PI/16

ATAN(R) = R - R*Q, $Z = R^2$

$$Q = \frac{n_0 + n_1 Z + n_2 Z^2 + n_3 Z^3}{d_0 + d_1 Z + d_2 Z^2 + d_3 Z^3}$$

n_0 = .13513 50000 00000 E + 06

n_1 = .21700 74603 93686 E + 06

n_2 = .97799 30329 54140 E + 05

n_3 = .10721 37452 05930 E + 05

d_0 = .17499 99999 99999 E - 11

d_1 = .45044 99999 99981 E + 05

d_2 = .45308 82013 16777 E + 05

d_3 = .85032 75632 14686 E + 04

Reference: 6400 FORTRAN Extended Library

ATAN2

Purpose: To compute the arctangent of the ratio of two real numbers.

Usage: B = ATAN2(Y,X)

Where X and Y are the single precision floating point arguments, and B is the single precision floating point result.

Normal Return: Return with result in B.

Error Messages: INDEFINITE ARGUMENT IN ATAN2
X = Y = 0.0

The message is written on the standard output file and displayed on the user's terminal; the result is set to indefinite and a normal exit is taken from ATAN2.

Storage: 98 words

Accuracy: 45 bits approximately

Mathematical Method: Let B = ATAN2(Y,X), then B is the argument of the complex number X + iY and $-PI \leqslant B \leqslant +PI$

$$B = \begin{cases} \text{sign (Y)*PI/2}, & X = 0 \\ \text{ATAN(Y/X)}, & X > 0 \\ \text{ATAN(Y/X) + sign (Y)*PI}, & X < 0 \end{cases}$$

Purpose: To compute the modulus of a complex number.

Usage: A = CABS(Z)

Where Z is the complex valued argument and A is the real result.

Normal Return: Return with result in A.

Error Messages: INDEFINITE ARGUMENT IN CABS
FLOATING OVERFLOW IN CABS

The message is written on the standard output file and displayed on the user's terminal; the result is set to indefinite and a normal exit is taken from CABS.

Storage: 32 words

Accuracy: 45 bits approximately

Mathematical Method: Let Z = X + iY, then

$$A = (X^2 + Y^2)^{1/2}$$

The square root function is evaluated by the machine instruction SQRT.

CCOS

Purpose: To compute the complex valued cosine of a complex valued number.

Usage: R = CCOS(Z)

Where Z is the complex valued argument and R is the complex valued result.

Normal Return: Return with result in R.

Error Messages:
INDEFINITE ARGUMENT IN CCOS
ABS (REAL PART) TOO LARGE IN CCOS
IMAG. PART TOO LARGE IN CCOS

The message is written on the standard output file and displayed on the user's terminal; the result is set to indefinite (both real and imaginary parts), and a normal exit is taken from CCOS.

Storage: 72 words (shared with CSIN)

Accuracy: 45 bits approximately

Mathematical Method: Let Z = X + iY, R = U + iV, then

$U = COS(X) * (e^{Y} + e^{-Y})/2$

$$V = \begin{cases} -SIN(X)*(e^{Y} - e^{-Y})/2 & \text{for abs (Y)} \geqslant 0.5 \\ -SIN(X)*Y*\sum_{n=0}^{5} c_n Y^{2n} & \text{for ab(Y)} < 0.5 \end{cases}$$

Where

C_0 = .99999 99999 99999 98116 72 E + 00
C_1 = .16666 66666 66672 12323 95 E + 00
C_2 = .83333 33333 07759 961 E - 02
C_3 = .19841 27027 90799 9 E - 03
C_4 = .27556 98073 56154 E - 05
C_5 = .25172 61882 51 E -07

The real valued sine, cosine, and exponential functions are evaluated as described in the respective routines.

If abs(X) > .110534964875444 E + 15 or if Y > 19905.80
the result is set to indefinite and the appropriate error message is issued.

Reference: "Computer Approximations", Hart, Cheyney, Lawson et al, John Wiley & Sons (New York) 1968 (Index SINH 1985).

Purpose: To compute the complex valued sine of a complex valued number.

Usage: R = CSIN(Z)

Where Z is the complex valued argument and R is the complex valued result.

Normal Return: Return with result in R.

Error Messages:
INDEFINITE ARGUMENT IN CSIN
ABS (REAL PART) TOO LARGE IN CSIN
IMAGINARY PART TOO LARGE IN CSIN

The message is written on the standard output file and displayed on the user's terminal; the result is set to indefinite (both real and imaginary parts), and a normal exit is taken from CSIN.

Storage: 72 words (shared with CCOS).

Accuracy: 45 bits approximately

Mathematical Method: Let $Z = X + iY$, $R = U + iV$, then $U = SIN(X)*(e^{Y} + e^{-Y})/2$

$U = SIN(X)*(e^{Y} + e^{-Y})/2$

$$V = \begin{cases} COS(X)*(e^{Y} - e^{-Y})/2 & \text{for } abs(Y) \geqslant 0.5 \\ COS(X)*Y*\sum_{n=0}^{5} C_n\, Y^{2n} & \text{for } abs(Y) < 0.5 \end{cases}$$

Where C_n are as given in routine CCOS.

Real valued sine, cosine and exponential functions are evaluated as described in the respective routines.

If abs(X) > .110534964875444 E + 15
or if Y > 19905.80
the result is set to indefinite and the appropriate error message is issued.

CEXP

Purpose: To compute the complex valued exponential of a complex valued number.

Usage: R = CEXP(Z)

Where Z is the complex valued argument and R is the complex valued result.

Normal Return: Return with result in R.

Error Messages: INDEFINITE ARGUMENT IN CEXP
REAL PART TOO LARGE IN CEXP
ABS (IMAG PART) TOO LARGE IN CEXP

The message is written on the standard output file and displayed on the user's terminal; the result is set to indefinite (both real and imaginary parts), and a normal exit is taken from CEXP.

Storage: 49 words

Accuracy: 45 bits approximately

Mathematical Method: Let Z = X + iY, R = U + iV

then

$$U = COS(Y)*e^X \qquad V = SIN(Y)*e^X$$

Real valued sine, cosine, and exponential functions are evaluated as described in the respective procedures.

If X > 19905.80 or if ABS(Y) > .110534964875444 E + 15

the result is set to indefinite and the appropriate error message is issued.

Purpose: To compute the complex valued logarithm of a complex valued number.

Usage: R = CLOG(Z)

Where Z is the complex valued argument and R is the complex valued result.

Normal Return: Return with result in R.

Error Messages:
INDEFINITE ARGUMENT IN CLOG
ZERO ARGUMENT IN CLOG
FLOATING OVERFLOW IN CLOG

The message is written on the standard output file and displayed on the user's terminal; the result is set to indefinite (both real and imaginary parts) and a normal exit is taken from CLOG.

Storage: 46 words

Accuracy: 45 bits approximately

Mathematical Method: Let $Z = X + iY$, $R = U + iV$

$$U = LOG_e((X^2 + y^2)^{1/2})$$

$$V = \text{arctangent } (Y/X)$$

The real valued log and arctangent functions are computed as described for the functions ALOG and ATAN2. The square root is computed by the machine instruction SQRT.

CSQRT

Purpose: To compute the complex valued square root of a complex valued argument.

Usage: R = CSQRT(Z)

Where Z is the complex valued argument and R is the complex valued result.

Normal Return: Return with result in R.

Error Messages: INDEFINITE ARGUMENT IN CSQRT
FLOATING OVERFLOW IN CSQRT

The message is written on the standard output file and displayed on the user's terminal; the result is set to indefinite (both real and imaginary parts) and a normal exit is taken from CSQRT.

Storage: 42 words

Accuracy: 45 bits approximately.

Mathematical Method: Let $Z = X + iY$, $R = U + iV$

$$A = (X^2 + Y^2)^{1/2}$$

$$B = (A + \text{abs}(X)/2)^{1/2}$$

$$C = \text{abs}(Y)/2B$$

If $X \geq 0$ $U = B$
$V = C * \text{sign}(Y)$

If $X < 0$ $U = C$
$V = B * \text{sign}(Y)$

If $X = 0$ and $Y = 0$, $U = 0$
$V = 0$

The square root function is computed by means of the machine instruction SQRT.

Purpose: To compute the exponential of a real number.

Usage: Y = EXP(X)

Where X is the single precision floating point argument, and Y is the result in single precision floating point.

Normal Return: Return with result in Y.

Error Messages: INDEFINITE ARGUMENT IN EXP
ARGUMENT TOO LARGE, FLOATING OVERFLOW IN EXP

The message is written on the standard output file and displayed on the user's terminal; the result is set to indefinite and a normal exit is taken from EXP.

Storage: 55 words

Accuracy: 45 bits approximately

Mathematical Method: $e^X = 2^K * 2^{L/16} * 2^{F/16}$, where

$K = [N/16]$ = integral part of bracketed quantity

if $X \geq 0$

$= [N/16] - 1$ if $X < 0$

$L = N$ modulo 16 if $X \geq 0$

$= 16 - (N$ modulo $16)$ $X < 0$

$N = [16 * (X/\log_e (2))]$

$F = (16 * (X/\log_e(2))) - N$ $(0 \leq \text{abs}(F) < 1)$

The factor $2^{L/16}$ is obtained by table lookup.

The product $2^K * 2^{L/16}$ is obtained by adding K to the exponent of $2^{L/16}$.

$2^{F/16} = (Q + F * P)/(Q - F * P)$

$Q = Q01 * F^2 + Q00$

$P = P01 * F^2 + P00$

Q00 = .53283 25426 30989 E + 4

Q01 = .1 E + 1

P00 = .11541 60545 73517 E + 3

P01 = .36100 70989 48762 E - 2

If $X < -19842.031$ the result is set to zero and a normal return is taken.

If X > 19905.80 the result is set to indefinite and the error message FLOATING OVERFLOW is issued.

Reference: "Computer Approximations", Hart, Cheyney, Lawson et al, John Wiley & Sons (New York), 1968 pp. 96-104.

6400 FORTRAN Extended Library

SIN

Purpose: To compute the sine of a real argument expressed in radians.

Usage: Y = SIN(X)

Where X is the single precision floating point argument, and Y is the result in single precision floating point.

Normal Return: Return with result in Y.

Error Messages: INDEFINITE ARGUMENT IN SIN
ARGUMENT TOO LARGE, ACCURACY LOST IN SIN

The message is written on the standard output file and displayed on the user's terminal; the result is set to indefinite and a normal exit is taken from SIN.

Storage: 77 words (shared with COS).

Accuracy: 45 bits approximately.

Mathematical Method: Let R = abs(X)*4/PI; N = [R] = integral part of R

T = R-N $(0 \leqslant T < 1)$

K = N modulo 8, K = 0, 1, 2, 3, 4, 5, 6, 7

If K = 0, SIN(X) = SIN(Z), Z = T

K = 1, SIN(X) = COS(Z), Z = 1 - T

K = 2, SIN(X) = COS(Z), Z = T

K = 3, SIN(X) = SIN(Z), Z = 1 - T

K = 4, SIN(X) = - SIN(Z), Z = T

K = 5, SIN(X) = - COS (Z), Z = 1 - T

K = 6, SIN(X) = - COS(Z), Z = T

K = 7, SIN(X) = - SIN(Z), Z = 1 - T

$$SIN(Z) = Z \sum_{n=0}^{6} S_n T^{2n}; \quad COS(Z) = \sum_{n=0}^{6} C_n T^{2n}$$

$S_0 =$ .78539 81633 97448 30701 4 D + 00

$S_1 = -$ .80745 51218 82805 30192 D - 01

$S_2 =$ .24903 94570 18873 6117 D - 02

$S_3 = -$ .36576 20415 84556 95 D - 04

$S_4 =$ .31336 16216 61904 D - 06

$S_5 = -$ .17571 49292 755 D - 08

$S_6 =$ .68771 00349 D - 11

$C_0 =$ .99999 99999 99999 94429 D + 00

$C_1 = -$ .30842 51375 34037 22987 D + 00

$C_2 =$ .15854 34424 37345 682 D - 01

$C_3 = -$ .32599 18864 54040 01 D - 03

$C_4 =$ .35908 59123 36036 D - 05

$C_5 = -$ .24609 45716 614 D - 07

$C_6 =$ .11363 81269 7 D - 09

If abs(X) > .110534964875444 E + 15, the result is set to indefinite and the appropriate error message is issued.

Reference: "Computer Approximations", Hart, Cheyney, Lawson et al, John Wiley & Sons (New York) 1968 p. 117 (INDICES SIN 3043, COS 3823).

COS

Purpose: To compute the cosine of a real argument expressed in radians.

Usage: Y = COS(X)

Where X is the single precision floating point argument, and Y is the result in single precision.

Normal Return: Return with result in Y.

Error Messages: INDEFINITE ARGUMENT IN COS
ARGUMENT TOO LARGE, ACCURACY LOST IN COS

The message is written on the standard output file and displayed on the user's terminal; the result is set to indefinite and a normal exit is taken from COS.

Storage: 77 words (shared with SIN).

Accuracy: 45 bits approximately.

Mathematical Method: Let R = abs(X)*4/PI; N = integer part (R)

$T = R - N \quad (0 \leqslant T < 1)$

K = (N + 2) modulo 8, K = 0, 1, 2, 3, 4, 5, 6, 7

Then COS(X) = SIN(X)

where SIN(X) is computed as described for each value of K in the description of the SIN function. If abs(X) > .110534964875444 E + 15, the result is set to indefinite, and the appropriate error message is issued.

Purpose: To compute the square root of a real number.

Usage: Y = SQRT(X)

Where X is the single precision floating point argument, and Y is the single precision floating point result.

Normal Return: Return with result in Y.

Error Messages: NEGATIVE ARGUMENT IN SQRT
INDEFINITE ARGUMENT IN SQRT

The message is written on the standard output file and displayed on the user's terminal; the result is set to indefinite and a normal exit is taken from SQRT.

Storage: 25 words

Accuracy: 45 bits approximately.

Mathematical Method: The square root function is computed by means of the machine instruction SQRT.

TAN

Purpose: To compute the tangent of a real number expressed in radians.

Usage: Y = TAN(X)

Where X is the single precision floating point argument, and Y is the single precision floating point result.

Normal Return: Return with result in Y.

Error Messages:

INDEFINITE ARGUMENT IN TAN
MAGNITUDE OF ARGUMENT IS TOO LARGE
ACCURACY LOST IN TAN
FLOATING OVERFLOW IN TAN

The message is written on the standard output file and displayed on the user's terminal; the result is set to indefinite and a normal exit is taken from TAN.

Storage: 98 words (shared with COTAN)

Accuracy: 45 bits approximately

Mathematical Method: Let R = abs(X*8/PI), N = [R] = integral part of R

$Z = R - N \qquad (0 \leqslant Z < 1)$

N = L modulo 8; $K = \begin{cases} L & \text{if } 0 \leqslant L \leqslant 3 \\ (7 - L) & \text{if } 4 \leqslant L \leqslant 7 \end{cases}$

If M is even (X reduces to interval [0, PI/2])

Then for

K = 0, TAN(X) = sign(X)*TAN(Y) where Y = Z

K = 1, TAN(X) = sign(X)*(1 - TAN(Y))/(1 + TAN(Y)), where y = 1 - Z

K = 2, TAN(X) = sign(X)*(1 + TAN(Y))/(1 - TAN(Y)), where Y = Z

K = 3, TAN(X) = sign(X)*(1/TAN(Y)) where Y = 1 -z

If M is odd (X reduces to interval [PI/2, PI])

Then for

K = 0, TAN(X) = - sign(X)*TAN(Y) where Y = 1 - Z

K = 1, TAN(X) = - sign(X)*(1 - TAN(Y))/(1 + TAN(Y)) where Y = Z

K = 2, TAN(X) = - sign(X)*(1 + TAN(Y))/(1 - TAN(Y)) where Y = 1- Z

K = 3, TAN(X) = - sign(X)*(1/TAN(Y)) where Y = Z

Where

$$TAN(Y) = Y \sum_{n=0}^{7} C_n y^{2n}$$

C_0 = .39269 90816 98721 2163 E + 0

C_1 = .20186 37804 74456 405 E - 1

C_2 = .12451 97277 23964 36 E - 2

C_3 = .77724 49638 61939 E - 4

C_4 = .48568 62790 1411 E - 5

C_5 = .30407 65954 617 E - 6

C_6 = .18381 90939 79 E - 7

C_7 = .15320 04254 E - 8

If X = PI/2, the error message FLOATING OVERFLOW is issued.

If abs(X) > (2^{47} - 1), the error message MAGNITUDE OF ARGUMENT TOO LARGE, ACCURACY LOST is issued.

Reference: "Computer Approximations", Hart, Cheyney, Lawson et al, John Wiley & Sons, (New York), 1964, (Index No. TAN 4186).

COTAN

Purpose: To compute the contangent of a real argument expressed in radians.

Usage: Y = COTAN(X)

Where X is the single precision floating point argument, and Y is the single precision floating point result.

Normal Return: Return with result in Y

Error Messages:

INDEFINITE ARGUMENT IN COTAN
MAGNITUDE OF ARGUMENT IS TOO LARGE,
ACCURACY LOST IN COTAN
FLOATING OVERFLOW IN COTAN

The message is written on the standard output file, and displayed on the user's terminal; the result is set to indefinite and a normal exit is taken from COTAN.

Storage: 98 words (shared with TAN)

Accuracy: 45 bits approximately.

Mathematical Method: Let $R = abs(X * 8/PI)$, $N = [R]$ = integral part of R

$Z = R - N$ $\quad (0 \leqslant Z < 1)$

$N = L$ modulo 8 $\quad K = \begin{cases} L & \text{if } 0 \leqslant L \leqslant 3 \\ (7 - L) & \text{if } 4 \leqslant L \leqslant 7 \end{cases}$

$M = [abs(X * 2/PI)]$ = integral part of bracketed quantity.

If M is even (X reduces to interval [0, PI/2]).

Then for

$K = 0$, $COTAN(X) = sign(X) * (1/TAN(Y))$ where $Y = Z$

$K = 1$, $COTAN(X) = sign(X) * (1 + TAN(Y))/(1 - TAN(Y))$ where $Y = 1 - Z$

$K = 2$, $COTAN(X) = sign(X) * (1 - TAN(Y))/(1 + TAN(Y))$ where $Y = Z$

$K = 3$, $COTAN(X) = sign(X) * TAN(Y)$ where $Y = 1 - Z$

If M is odd (X reduces to interval [PI/2, PI])

Then for

$K = 0$, $COTAN(X) = - sign(X) * (1/TAN(Y))$ where $Y = 1 - Z$

$K = 1$, $COTAN(X) = - sign(X) * (1 + TAN(Y))/(1 - TAN(Y))$ where $Y = Z$

$K = 2$, $COTAN(X) = - sign(X) * (1 - TAN(Y))/(1 + TAN(Y))$ where $Y = 1 - Z$

$K = 3$, $COTAN(X) = - sign(X) * TAN(Y)$ where $Y = Z$

where

$$TAN(Y) = Y \sum_{n=0}^{7} C_n Y^{2n}$$

where C_n are constants defined in description of function TAN.

If X = 0, the error message FLOATING OVERFLOW is issued.

If $abs(X) > (2^{47} - 1)$, the error message MAGNITUDE OF ARGUMENT TOO LARGE, ACCURACY LOST is issued.

Purpose: To compute the hyperbolic tangent of a real argument expressed in radians.

Usage: Y = TANH(X)

Where X is the single precision floating point argument, and Y is the single precision floating point result.

Normal Return: Return with result in Y.

Error Messages: INDEFINITE ARGUMENT IN TANH

The message is written on the standard output file, and displayed on the user's terminal; the result is set to indefinite and a normal exit is taken from TANH.

Storage: 48 words.

Accuracy: 45 bits approximately.

Mathematical Method: For $0 \leqslant abs(X) \leqslant .12$

$$TANH(X) = X \sum_{n=0}^{5} C_n X^{2n}$$

Where

$C_0 = 1$

$C_1 = -1/3$

$C_2 = 2/15$

$C_3 = -17/315$

$C_4 = 62/2835$

$C_5 = -1382/155925$

for $.12 < abs(X) \leqslant 18.0$

$$TAHN(X) = (e^X - e^{-X})/(e^X + e^{-X})$$

$$= 1 - (2/(e^{2X} + 1))$$

Where the exponential function is computed as described for function EXP.

for $abs(X) > 18.0$

$$TANH(X) = sign(X) * 1.0$$

Reference: 6400 FORTRAN Extended Library.

DATAN
DATAN2

Purpose: To compute in double precision the arctangent of a double precision number, or the arctangent of the ratio of two numbers.

Usage: Z = DATAN(Y) or Z = DATAN2(Y, X)

Where X and Y are double precision floating point arguments, and Z is the result in double precision.

Normal Return: Return with result in Z.

Error Messages: INDEFINITE ARGUMENT IN DATAN (DATAN2)
X = Y = 0.0 IN DATAN (DATAN2)

If any of the above error conditions occur, the message is written on the standard output file and displayed on the user's terminal; the result is set to indefinite and a normal exit is taken fron DATAN.

Storage: 235 words.

Accuracy: 90 bits approximately.

Mathematical Method: For DATAN(Y) let X = 1 then DATAN(Y) = DATAN2(Y, X)

Let Z = MIN(ABS(X), ABS(Y))/MAX(ABS(X), ABS(Y))

Let V = ABS(Y/X)

DATAN(V) = DATAN(Z) if ABS(Y) .LE. ABS(X)
= PI/2 - DATAN(Z) if ABS(X) .LT. ABS(Y)

Where DATAN(Z) is calculated as follows

Let P = TAN(PI/16), T = TAN(3PI/16)

DATAN(Z) = DATAN(R) + C R and C defined below

O.LE.Z.LT.P	R = Z	C = 0
O.LE.Z.LT.SQRT(2) - 1	R = (Z - P)/(1 + Z * P)	C = PI/16
SQRT(2) - 1.LE.Z.LT.1	R = (Z - T)/(1 + Z * T)	C = 3PI/16
1.LE.Z.LT.SQRT(2) + 1	R = (1 - Z * T)/(Z + T)	C = 5PI/16
SQRT(2) + 1.LE.Z	R = (1 - Z * P)/(Z + P)	C = 7PI/16)

where DATAN(R) is computed from a telescoped Taylor-Maclauren Power Series.

DATAN2(Y/X)	= SIGN(Y)*PI/2	if X = 0	
	= DATAN(V)*SIGN(Y)	X > 0	
	= PI - DATAN(V)	X < 0 and Y.GE.0	
	= DATAN(V) - PI	X < 0 and Y < 0	

References 6400 FORTRAN Extended Library

DEXP

Purpose: To compute the exponential of a double precision number.

Usage: Y = DEXP(X)

Where X is the double precision floating point argument, and Y is the result in double precision.

Normal Return: Return with result in Y.

Error Messages: INDEFINITE ARGUMENT IN DEXP
ARGUMENT TOO LARGE, FLOATING OVERFLOW IN DEXP

The message is written on the standard output file and displayed on the user's terminal; the result is set to indefinite and a normal exit is taken from DEXP.

Storage: 120 words.

Accuracy: 90 bits approximately.

Mathematical Method: Let N = [X/LN(2) + .5]
and R = R1 + R2 = X - N*LN(2), ABS(R) < = LN(2)

R1 is the most significant part of R

R2 is the least significant part of R

E ** R1 is evaluated from a polynomial of degree 17.

The polynomial was telescoped from a truncated Malclauren Power Series.

E ** R2 = (1 + R2)

E ** X = (2 ** N)*(E ** R1)*(E ** R2)

References: 6400 FORTRAN Extended Library.

DLOG

Purpose: To compute the double precision logarithm to base e of a double precision number.

Usage: Y = DLOG(X)

Where X is the double precision floating point argument, and Y is the result in double precision.

Normal Return: Return with result in Y.

Error Messages:
INDEFINITE ARGUMENT IN DLOG
ZERO ARGUMENT IN DLOG
NEGATIVE ARGUMENT IN DLOG

The message is written on the standard output file and displayed on the user's terminal; the result is set to indefinite and a normal exit is taken from DLOG.

Storage: 196 words (shared with DLOG10).

Accuracy: 90 bits approximately.

Mathematical Method: DLOG(X) is computed as follows:

X = (2**K)*w where SQRT(1/2) < = w < SQRT(2)

then DLOG(X) = K*LOG(2) + LOG(W)

LOG(W) is approximated by

C1*T + C3*T**3 + C5*T**5 + C7*T**7, where T = (W - 1)(W + 1)

The iteration formula for F(A) = E**A - X = 0 is

A(N + 1) = A(N) - (1 - X*E** - A(N))

Let R = X*E** - A0 and T = 1 - R

R1, T1, R2, T2 denote the 2 significant parts of T and R

The final result with desired accuracy is:

A2 = A0 - T1 - T2 - (T1**2)*(1/2 + T1/2)

References: 6400 FORTRAN Extended Library

Purpose: To compute the double precision logarithm to base 10 of a double precision number.

Usage: Y = DLOG10(X)

Where X is the double precision floating point argument, and Y is the result in double precision.

Normal Return: Return with result in Y.

Error Messages:
INDEFINITE ARGUMENT IN DLOG10
ZERO ARGUMENT IN DLOG10
NEGATIVE ARGUMENT IN DLOG10

The message is written on the standard output file and displayed on the user's terminal; the result is set to indefinite and a normal exit is taken from DLOG10.

Storage: 196 words (shared with DLOG).

Accuracy: 90 bits approximately.

Mathematical Method: DLOG10(X) = LOG BASE10 (E) * DLOG(X)

Where DLOG(X) is computed as in DLOG routine.

DSINCOS

Purpose: To compute the double precision sine or cosine for a double precision number expressed in radians.

Usage: Y = DSIN(X) or Y = DCOS(X)

Where X is the double precision floating point argument, and Y is the result in double precision.

Normal Return: Return with result in Y.

Error Messages: INDEFINITE ARGUMENT IN DSIN(DCOS)
ARGUMENT TOO LARGE IN DSIN(DCOS)

The message is written on the standard output file, and is displayed on the user's terminal; the result is set to indefinite and a normal exit is taken from DSIN (DCOS).

Storage: 160 words.

Accuracy: 90 bits approximately.

Mathematical Method: Let N = INT(ABS(X)*2/PI + .5)

Let R = X - N*PI/2 then ABS(R).LE.PI/4

Let K = ABS(N) mod 4, K = 0, 1, 2, 3

then SIN(X) = SIN(R)*COS(K*PI/2) + COS(R)*SIN(K*PI/2) and

COS(X) = SIN(R)*SIN(K*PI/2) - COS(R)*COS(K*PI/2)

Depending upon whether SIN(X) or COS(X) is wanted and upon the value of K, either the SIN or COS of R is evaluated and complemented if necessary.

The SIN and COS of R are evaluated by polynomials of degree 21 and 20 respectively. These polynomials were telescoped from a truncated Taylor-Maclauren Power Series of degree 25 and 24.

References: 6400 FORTRAN Extended Library.

Purpose: To compute the double precision square root of a double precision number.

Usage: Y = DSQRT(X)

Where X is the double precision floating point argument, and Y is the result in double precision.

Normal Return: Return with result in Y.

Error Messages: INDEFINITE ARGUMENT IN DSQRT
NEGATIVE ARGUMENT IN DSQRT

The message is written on the standard output file and is displayed on the user's terminal; the result is set to indefinite and a normal exit is taken from DSQRT.

Storage: 46 words.

Accuracy: 90 bits approximately.

Mathematical Method: An approximation to the square root is obtained using the SQRT instruction. This number is accurate to 14 decimal places. One Newton approximation is done to double the accuracy of the number; the form is

$$A2 = 1/2*(A1 + X/A1)$$

DTAN

Purpose: To compute the tangent of a double precision number.

Usage: Y = DTAN(X)

Where X is a double precision floating point argument, and Y is the result in double precision.

Normal Return: Return with result in Y.

Error Messages: INDEFINITE ARGUMENT IN DTAN
ARGUMENT TOO LARGE IN DTAN

The message is written on the standard output file and displayed on the user's terminal; the result is set to indefinite and a normal exit is taken from DTAN.

Storage: 160 words.

Accuracy: 90 bits approximately.

Mathematical Methods: Let N = INT[X*2/PI + .5]

Let R = X - N*PI/2 Then ABS(R).LE.PI/4

Let K = ABS(N) MOD4, K = 0, 1, 2, 3

Then TAN(X) = SIN(X)/COS(X) where

SIN(X) = SIN(R)*COS(K*PI/2) + COS(R)*SIN(K*PI/2)

COS(X) = SIN(R)*SIN(K*PI/2) - COS(R)*COS(K*PI/2)

The SIN and COS of R are evaluated by polynomials telescoped from truncated Taylor-Maclauren Power Series.

Reference: MODIFICATION OF DISINCOS ROUTINE

ERROR DIAGNOSTICS D

Error diagnostics are produced when the compiler detects FORTRAN syntax errors in the source program or when the source program gives the compiler illegal commands. The seriousness of the error is indicated by the error type code:

W	(Warning)	The statement in error was compiled. Compilation continued. At object time the run executed.
F	(Fatal)	The statement in error was not compiled. Compilation did not continue. At object time the run was terminated.

Error diagnostics are produced also when the compiler fails. The error type code for compiler failure is:

A	(Abort)	Compilation was terminated and object time execution was terminated.

Error Number	Type	Message
93	A	COMPILER FAILURE - SUBSCRIPT REFERENCE FOR NON-DIMENSIONED ARRAY Subscript processor has detected a bad symbol table entry.
94	A	COMPILER FAILURE - ALL FULL REG TABLE ENTRIES ARE CLOSE 4 (GFFULLRG) The full word register assignment table in generation phase has gone bad.
95	A	COMPILER FAILURE - ALL HALF REG TABLE ENTRIES ARE CALL 4 (GFHALFRG) The half word register assignment table in generation phase has gone bad.

ERROR Number	Type	Message
96	A	COMPILER FAILURE - VARIABLE EQUIVALENCED TO COMMON THAT HAS NO ELEMENTS Storage class table has gone bad in allocation phase.
97	A	COMPILER FAILURE - CANNOT FIND SYMBOL IN SYMBOL TABLE The symbol table has gone bad.
98	A	COMPILER FAILURE - I/O STACK FORMED INCORRECTLY I/O list stack that was built by IOLIST processor has gone bad in parse phase.
99	A	COMPILER FAILURE - ILLEGAL DESCRIPTOR ENCOUNTERED IN ALLOCATION PHASE (2) Descriptor table has gone bad.
100	A	COMPILER FAILURE - TABLE AREA OVERFLOW One of the 8 table areas has reached its maximum size. Either the program is too big to be compiled or something is wrong in the compiler.
101	A	COMPILER FAILURE Something has gone wrong in the compiler.
102	F	ILLEGAL SUBPROGRAM NAME Subprogram is compiled as a main program.
103	F	FUNCTION CANNOT BE CALLED AS A SUBROUTINE
104	W	CANNOT TYPE SUBROUTINE NAME It has no meaning to type the name.
105	F	ILLEGAL SUBROUTINE REFERENCE

ERROR Number	Type	Message
106	F	MISSING OPERATOR OR DELIMITER
107	F	ILLEGAL OPERAND
108	F	ILLEGAL OR MISSING DELIMITER
109	F	ILLEGAL USE OF ARRAY NAME Array name must be subscripted.
110	F	MISSING LEFT PARENTHESIS (
111	F	ILLEGAL USE OF STATEMENT FUNCTION ARGUMENT
112	F	RECURSIVE SUBPROGRAM REFERENCE IS ILLEGAL A subprogram may not call itself.
113	F	ILLEGAL ARGUMENT DELIMITER
114	F	ILLEGAL USE OF FUNCTION NAME
115	F	ILLEGAL ARGUMENT IN INTRINSIC OR BASIC FUNCTION REFERENCE The arguments are not what the function expects.
116	W	FUNCTION NAME USED OR ARGUMENT NOT DECLARED EXTERNAL
117	F	INTRINSIC FUNCTION CANNOT BE ACTUAL ARGUMENT
118	F	ILLEGAL DELIMITER PAIR
119	F	PARENTHESES DO NOT MATCH There is not a one to one correspondence between left and right parentheses.

ERROR Number	Type	Message
120	F	INCORRECT NUMBER OF ARGUMENTS FOR INTRINSIC OR BASIC FUNCTION
121	F	INCORRECT ARGUMENT TYPE FOR INTRINSIC OR BASIC FUNCTION
122	F	ILLEGAL TYPE MIXING IN STATEMENT
124	F	ILLEGAL MODE USAGE OF RELATIONAL EXPRESSIONS
125	W	MORE THAN 19 CONTINUATION LINES All continuation lines after 19 are ignored.
126	W	THIS STATEMENT CANNOT BE EXECUTED The statement before this one will not allow execution of this statement.
127	W	INDEFINITE RESULT PRODUCT TOO LARGE
128	W	DIVIDE FAULT IN CONSTANT ARITHMETIC The division of one constant by another has produced a divide fault.
129	W	EXPONENT OVERFLOW IN CONSTANT ARITHMETIC The multiplication of two constants has produced exponent overflow.
131	F	STATEMENT FUNCTION DEFINITION MUST PRECEDE ALL EXECUTABLE STATEMENTS The statement looks like a statement function definition.
132	F	THIS SYMBOL MAY NOT BE DEFINED TO BE A STATEMENT FUNCTION The symbol is already defined.

ERROR Number	Type	Message
133	F	ILLEGAL STATEMENT FUNCTION ARGUMENT
134	F	ILLEGAL STATEMENT FUNCTION DEFINITION
135	F	ILLEGAL LABEL
136	F	I/O STATEMENT REFERS TO NON-FORMAT STATEMENT
137	F	ILLEGAL REFERENCE TO FORMAT
138	F	DOUBLY DEFINED LABEL
139	F	INCORRECT ARGUMENT TYPE FOR STATEMENT FUNCTION The actual argument does not agree in type with the dummy argument.
140	F	ILLEGAL DELIMITER IN STATEMENT FUNCTION ARGUMENT LIST
141	F	INCORRECT NUMBER OF ARGUMENTS FOR STATEMENT FUNCTIONS
142	F	COMPLEX MAY NOT BE USED AS EXPONENT
143	F	COMPLEX MAY ONLY HAVE EXPONENT OF INTEGER OR REAL
144	F	SUBSCRIPT MUST BE INTEGER CONSTANT
145	F	SPECIFICATION STATEMENTS MUST PRECEDE ALL EXECUTABLE STATEMENTS
146	F	ILLEGAL VARIABLE IN DATA STATEMENT The symbol is defined to be something that cannot be preset.

ERROR Number	Type	Message
147	F	SYNTAX ERROR IN DATA LIST
148	F	SUBSCRIPT MAY NOT BE AN EXPRESSION
149	F	TOO MANY SUBSCRIPTS
150	F	SYNTAX ERROR IN HEXADECIMAL CONSTANT
151	F	ILLEGAL DATA ITEM
152	F	ARRAY MUST BE LAST ITEM TO BE INITIALIZED BY HEX CONSTANT
153	F	CHARACTER CONSTANT TOO LARGE
154	F	ARRAY MUST BE LAST ITEM TO BE INITIALIZED WITH CHARACTER CONSTANT
155	W	TOO MANY DATA CONSTANTS The excess constants are ignored
156	F	SYNTAX ERROR
157	F	SPECIFICATION STATEMENTS MUST PRECEDE STATEMENT FUNCTION DEFINITION
158	F	ILLEGAL OPERATOR IN SPECIFICATION LIST
159	F	ILLEGAL OPERATOR IN SPECIFICATION LIST
160	F	LENGTH SPECIFICATION OF CHARACTER MUST BE INTEGER CONSTANT
161	W	NAMELIST NAME IN TYPE STATEMENT It has no meaning to type a NAMELIST name.

ERROR Number	Type	Message
162	W	VARIABLE TYPED MORE THAN ONCE First type is retained.
163	F	LENGTH OF ADJUSTABLE CHARACTER MUST BE TYPED INTEGER
164	F	ZERO LENGTH FOR CHARACTER VARIABLE
165	F	MISSING , or *
166	F	ILLEGAL STATEMENT ON LOGICAL IF The logical IF part of statement was compiled.
167	W	NO LABELED COMMON IN BLOCK DATA SUBPROGRAM
168	F	ILLEGAL STATEMENT IN BLOCK DATA SUBPROGRAM
169	W	MAIN PROGRAM HAS NO EXECUTABLE STATEMENTS
170	W	NO STOP STATEMENT IN MAIN PROGRAM A STOP was generated.
171	W	END NOT PRECEDED BY BRANCH STATEMENT A STOP was generated.
172	W	FUNCTION NAME IS NOT DEFINED The function must take on a value during the execution of the subprogram.
173	W	NO RETURN STATEMENT A RETURN was generated.
174	F	ENTRY IN RANGE OF DO LOOP

ERROR Number	Type	Message
175	F	NO ARGUMENTS FOR FUNCTION The subprogram was compiled as a main program.
176	W	ILLEGAL DUMMY ARGUMENT The subprogram was compiled as a function or subroutine.
177	F	MISSING NAMELIST NAME
178	F	ILLEGAL NAMELIST NAME
179	F	MISSING SLASH AFTER NAMELIST NAME
180	F	LIST ITEM MUST BE A VARIABLE
181	F	ILLEGAL OPERATOR
182	F	ILLEGAL OR MISSING VARIABLE
183	F	SYNTAX ERROR IN LABEL STRING
184	F	ILLEGAL KEYPOINT VALUE
185	F	INVALID LABEL REFERENCE
186	F	MORE THAN 253 COMMON BLOCK NAMES
187	F	ATTEMPTED TO RE-ORDER COMMON
188	F	MORE THAN ONE ELEMENT OF A SET IN COMMON Two variables in COMMON can not be equivalenced.

ERROR Number	Type	Message
189	F	ENTRY MUST BE IN A SUBROUTINE OR FUNCTION
190	W	DUPLICATION OF DUMMY ARGUMENT NAMES The subprogram was compiled as a function or subroutine.
191	F	ILLEGAL DIMENSION SPECIFICATION
192	F	ILLEGAL FORMATION OF I/O STATEMENT
193	F	ILLEGAL ELEMENT IN UNIT POSITION
194	W	DUPLICATE OPTION IN I/O STATEMENT First option is retained.
195	F	ILLEGAL OPTION IN I/O STATEMENT
196	W	REFERENCED UNDEFINED FORMAT A FORMAT statement was supplied by the compiler.
197	F	ILLEGAL OR MISSING RECORD AREA PARAMETER
198	F	NO FORMAT REFERENCE
199	F	ILLEGAL ELEMENT IN I/O LIST
200	F	ILLEGAL OR MISSING DELIMITER IN I/O LIST
201	F	ILLEGAL FORMATION OF REWIND, ENDFILE OR BACKSPACE
202	F	ILLEGAL FORMATION OF COMMON STATEMENT
203	F	COMMON BLOCK NAME IS NOT SYMBOLIC

ERROR Number	Type	Message
204	F	DUPLICATE SYMBOLIC NAME IN COMMON STATEMENT
205	W	DATA SHOULD NOT BE PRESET IN BLANK COMMON
206	F	DUMMY ARGUMENT CANNOT APPEAR IN COMMON
207	F	ILLEGAL USE OF VARIABLE OR VARIABLE DIMENSIONED MORE THAN ONCE
208	F	A VARIABLE IN A DIMENSION STATEMENT MUST BE DIMENSIONED
209	F	MISSING COMMA
210	F	DIMENSIONING FORMAT ERROR
211	F	ILLEGAL USE OF SUBSCRIPT
212	F	VARIABLE DIMENSION WAS NOT A DUMMY ARGUMENT
213	F	VARIABLE DIMENSION HAS TO BE A SIMPLE VARIABLE
214	F	VARIABLE DIMENSION CANNOT BE DEFINED Subscript for variable dimensioned arrays cannot be changed.
215	F	MORE THAN 7 DIMENSIONS SPECIFIED
216	F	CONSTANT GREATER THAN 2**18 IN SPECIFICATION STATEMENT
217	F	ILLEGAL OR MISSING LABEL REFERENCE IN DO STATEMENT
218	F	LABEL REFERENCED GREATER THAN 99999
219	F	ILLEGAL PARAMETER IN DO STATEMENT

ERROR Number	Type	Message
220	F	ILLEGAL OR MISSING DELIMITER
221	W	END OCCURS BEFORE ALL DO LOOPS HAVE BEEN TERMINATED The compiler has supplied closing loop labels.
222	F	A DO LOOP MAY NOT TERMINATE ON THIS STATEMENT
223	F	EQUIVALENCE FORMAT ERROR
224	F	ILLEGAL COMPONENT BEING EQUIVALENCED
225	F	ILLEGAL DELIMITER SEPARATING EQUIVALENCE GROUPS
226	F	ARRAY ELEMENT MUST HAVE AT LEAST ONE SUBSCRIPT
227	F	ONLY SYMBOLIC NAMES CAN APPEAR IN EXTERNAL STATEMENTS
228	F	EXTERNAL STATEMENTS DID NOT PRECEDE REFERENCE OR VARIABLE IS WRONG TYPE
229	F	ILLEGAL USE OF NAME IN EXTERNAL STATEMENT
230	F	COMPLEX OR CHARACTER TYPE NOT ALLOWED IN ARITHMETIC IF
231	F	COMMA IS ONLY OPERATOR ALLOWED BETWEEN LABELS
232	F	SUBSCRIPT EXPRESSION NOT INTEGER, REAL OR DOUBLE PRECISION
233	F	I/O SPECIAL EXIT PARAMETER MUST BE AN INTEGER VARIABLE
234	F	ITEMS IN COMMON MUST BE ARRAYS OR SIMPLE VARIABLES
9935	F	END IS ILLEGAL IN DIRECT ACCESS I/O

ERROR Number	Type	Message
236	W	UNREFERENCED FORMAT
237	F	NAMELIST IS USED ILLEGALLY
238	W	UNREFERENCED NAMELIST
239	F	ADJUSTABLE LENGTH IS NOT A DUMMY ARGUMENT OR IN COMMON
240	F	INCORRECT DO SPECIFICATION IN I/O LIST
241	F	VARIABLE APPEARS IN COMMON MORE THAN ONCE
242	F	EQUIVALENCE RELATION ERROR BETWEEN GROUPS
243	F	NON-REDEFINABLE VARIABLE IN INPUT LIST
244	F	ARRAY REFERENCED WITH WRONG NUMBER OF SUBSCRIPTS
245	W	CONSTANT MAY BE TOO LARGE
246	F	EQUIVALENCE HAS ATTEMPTED TO REORIGIN COMMON
247	W	MISSING SUBSCRIPT - A ONE IS SUBSTITUTED
248	F	ILLEGAL COMPONENT IN I/O STATEMENT
249	F	ILLEGAL OR MISSING BUFFER SPECIFICATION
250	W	RETURN STATEMENT IGNORED IN BLOCK DATA SUBPROGRAM
251	W	RETURN STATEMENT REPLACED BY STOP STATEMENT IN MAIN PROGRAM
252	W	ILLEGAL PARAMETER IN RETURN STATEMENT

ERROR Number	Type	Message
253	W	MODE OF RETURN PARAMETER MUST BE INTEGER
254	W	ILLEGAL VALUE FOR RETURN STATEMENT
255	F	SYNTAX ERROR ON LEFT SIDE OF ASSIGNMENT STATEMENT
256	F	NON-REDEFINABLE VARIABLE ON LEFT SIDE OF ASSIGNMENT STATEMENT
257	F	ILLEGAL FIELD SPECIFICATION IN FORMAT
258	F	FORMAT STATEMENT IN BLOCK DATA SUBPROGRAM
259	F	LENGTH OF HOLLERITH FIELD OUT OF RANGE
260	F	END OF STATEMENT IN HOLLERITH FIELD The end of the record was reached before the N was satisfied.
261	F	MISSING CLOSING APOSTROPHE ON CHARACTER STRING
262	F	NO LABEL SPECIFIED IN ASSIGN STATEMENT
263	F	ASSIGN VARIABLE MUST BE SIMPLE INTEGER VARIABLE
264	F	MISSING SUBSCRIPTS
265	F	HEX CONSTANT TOO LARGE
266	F	ILLEGAL LABEL VALUE IN ASSIGN STATEMENT
267	F	ATTEMPT TO INITIALIZE CHARACTER VARIABLE WITH NON-CHARACTER DATA

ERROR Number	Type	Message
268	F	LOGICAL CONSTANT CAN NOT INITIALIZE OTHER TYPES
269	F	MISSING DATA The list of variables is longer than the list of data.
270	F	FLOATING POINT NUMBER OUT OF ALLOWABLE RANGE
271	F	ARRAY CANNOT BE PARTIALLY HEX OR CHARACTER
272	F	ATTEMPT TO REINITIALIZE VARIABLE
273	W	MISSING END STATMENT The compiler supplied an END statement.
274	F	ARRAY DECLARATOR NOT A VARIABLE
275	F	VARIABLE CANNOT BE DIMENSIONED
276	F	ATTEMPT TO REDIMENSION A VARIABLE
277	F	PROGRAM STARTS WITH A CONTINUATION CARD
278	F	SUBSCRIPT CANNOT BE ZERO
279	F	ARRAY HAS TO BE FORMAL ARGUMENT TO HAVE VARIABLE DIMENSION
280	F	VARIABLE DIMENSION SHOULD BE SIMPLE INTEGER VARIALBE
281	F	LOGICAL VARIABLE CANNOT BE INITIALIZED BY OTHER TYPES
283	W	EQUIVALENCE VARIABLE ATTEMPTED TO BE ASSIGNED TO IMPROPER BOUNDARY Compiler put variable on proper boundary.

ERROR Number	Type	Message
284	F	ILLEGAL ELEMENT IN ARGUMENT VECTOR
285	W	ILLEGAL FLOW IN THE PROGRAM
286	W	ILLEGAL TRANSFER INTO RANGE OF DO LOOP
287	W	REFERENCE TO UNDEFINED LABEL
288	W	ILLEGAL EXPONENTIATION
299	F	(-CONSTANT) ** (REAL OR DOUBLE PRECISION) IS ILLEGAL
300	W	EXTRANEOUS INFORMATION AT END OF STATEMENT
301	F	STATEMENT CANNOT BE IDENTIFIED
302	F	A LABEL MUST BE AN INTEGER CONSTANT
303	F	DIGIT STRING EXCEEDS MAXIMUM OF FIVE
304	F	ILLEGAL CHARACTER
305	W	ILLEGAL CONSTANT ON A PAUSE OR STOP
306	F	ILLEGAL CONSTANT TYPE
307	F	CHARACTER STRING EXCEEDS 255
308	F	HOLLERITH FIELD COUNT IS TOO LARGE
309	F	SYMBOLIC NAME HAS MORE THAN 8 CHARACTERS
310	F	COMPONENT HAS MORE THAN 255 CHARACTERS

ERROR Number	Type	Message
311	F	REAL NUMBER HAS MORE THAN 255 DIGITS
312	F	LOGICAL CONSTANT OR LOGICAL/RELATIONAL OPERATOR IS INCORRECT
313	F	ERROR IN HOLLERITH COUNT
314	F	REAL NUMBER CANNOT BE FOLLOWED BY A LETTER
315	F	COMPLEX NUMBER COMPONENTS CANNOT BE DOUBLE PRECISION
316	F	MISSING RIGHT PARENTHESIS)
317	F	SYNTAX ERROR IN A COMPLEX CONSTANT
318	F	ZERO LENGTH CHARACTER STRING
319	F	ILLEGAL ARGUMENT FIELD SYNTAX
320	W	IMPLICIT STATEMENT MUST BE FIRST SPECIFICATION STATEMENT
321	F	ILLEGAL TYPE IN IMPLICIT STATEMENT
322	F	ILLEGAL USE OF *
323	F	IMPLICIT RANGE IS INCORRECT
324	F	NON-FORTRAN CHARACTER ___ FOUND AND IS NOT IN HOLLERITH/CHARACTER STRING
325	F	SYNTAX ERROR AFTER A SYMBOLIC NAME
326	F	ILLEGAL CHARACTER AFTER A ZERO
327	F	SYNTAX ERROR AFTER AN INTEGER CONSTANT

ERROR Number	Type	Message
328	F	SYNTAX ERROR FOLLOWING A PERIOD
329	F	ILLEGAL CHARACTER IN A LOGICAL CONSTANT OR LOGICAL/RELATIONAL OPERATOR
330	F	SYNTAX ERROR FOLLOWING A REAL NUMBER
331	F	ILLEGAL CHARACTER APPEARS IN THE NUMBER PART OF THE EXPONENT FIELD
332	W	TOO MANY DIGITS IN THE EXPONENT FIELD
333	F	SYNTAX ERROR FOLLOWING A SYMBOLIC STRING THAT WAS FOLLOWED BY A PERIOD
334	F	SYNTAX ERROR FOLLOWING A LOGICAL CONSTANT
335	F	SYNTAX ERROR FOLLOWING A REAL CONSTANT
336	F	SYNTAX ERROR FOLLOWING AN *
337	F	SYNTAX ERROR FOLLOWING A CHARACTER STRING
338	F	SYNTAX ERROR FOLLOWING A COMPLEX CONSTANT
339	F	SYNTAX ERROR IN A LABEL REFERENCE FIELD
340	W	SUBSCRIPT REFERENCE OUT OF RANGE
341	F	DO LOOPS ARE NESTED ILLEGALLY
342	F	INDUCTION VARIABLE USED ILLEGALLY

INDEX

COMMENT SHEET

CONTROL DATA CORPORATION

TITLE: STAR Computer System FORTRAN Language Reference Manual

PUBLICATION NO. 60386200 REVISION A

This form is not intended to be used as an order blank. Control Data Corporation solicits your comments about this manual with a view to improving its usefulness in later editions.

Applications for which you use this manual.

Do you find it adequate for your purpose?

What improvements to this manual do you recommend to better serve your purpose?

Note specific errors discovered (please include page number reference).

CUT ON THIS LINE

General comments:

FROM NAME: _____ POSITION: _____

COMPANY NAME: _____

ADDRESS: _____

NO POSTAGE STAMP NECESSARY IF MAILED IN U.S.A.

FOLD ON DOTTED LINES AND STAPLE

STAPLE

STAPLE

FOLD

FOLD

FIRST CLASS
PERMIT NO. 8241

MINNEAPOLIS, MINN.

BUSINESS REPLY MAIL
NO POSTAGE STAMP NECESSARY IF MAILED IN U.S.A.

POSTAGE WILL BE PAID BY

CONTROL DATA CORPORATION
Documentation Department
215 Moffett Park Drive
Sunnyvale, California 94086

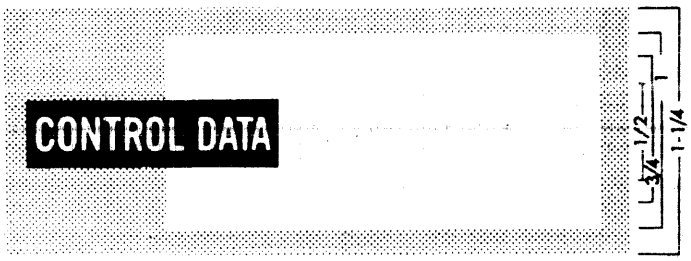
CUT ON THIS LINE

FOLD

FOLD

STAPLE

STAPLE



► ►CUT OUT FOR USE AS LOOSE-LEAF BINDER TITLE TAB

CONTROL DATA
CORPORATION

CORPORATE HEADQUARTERS, 8100 34th AVE. SO., MINNEAPOLIS, MINN. 55440
SALES OFFICES AND SERVICE CENTERS IN MAJOR CITIES THROUGHOUT THE WORLD

Pub. No. 60386200

Litho in U.S.A.