

Q U I C K R E F E R E N C E

CONVEX FORTRAN Quick Reference

Order No. DSW-039

Third Edition
November 1992



CONVEX

CONVEX COMPUTER CORPORATION

CONVEX FORTRAN

Quick Reference

Order No. DSW-039
Third Edition

Copyright © 1992 CONVEX Computer Corporation
All rights reserved.

CONVEX and the CONVEX logo ("C") are registered trademarks of
CONVEX Computer Corporation.

UNIX is a registered trademark of UNIX System Laboratories, Inc.

Compilation

The command line that invokes the CONVEX FORTRAN compiler is:

```
fc [options] files [loader-options]
```

where

options

is one or more compiler options (see below).

files

is one or more FORTRAN source files, object files, assembly files, or libraries.

loader-options

is one or more loader options.

Language-compatibility options

-cfc

Select Cray FORTRAN language definitions and library.

-cfcwpa

Compute word addresses when performing Cray pointer arithmetic on explicitly declared Cray pointers. Must be used with -cfc.

-dfc

Select DEC FORTRAN language definitions.

-F66

Select FORTRAN 66 language definitions.

-f90

Enable limited FORTRAN 90 compliance.

-sa

Prevent compiler from generating precompiled argument packets.

-sfc

Select Sun FORTRAN language definitions.

-vfc

Select VAX FORTRAN language definitions.

Optimization options

- ds
Perform dynamic loop selection.
- ep *n*
Specify estimated number of processors.
- il
Prepare intermediate-language file for inline substitution.
- is *directory*
Perform inline substitution.
- no
Perform no optimization.
- nopeel
Disable loop boundary value peeling.
- noptst
Disable test promotion.
- O0
Perform local scalar optimization.
- O1
Perform global scalar optimization.
- O2
Perform vectorization.
- O3
Perform automatic parallelization.
- peel
Increase code expansion limit for loop value boundary peeling.
- peelall
Remove code expansion limit for loop value boundary peeling.
- ptst
Increase code expansion limit for test promotion.
- ptstall
Remove code expansion limit for test promotion.

- rl
Perform loop replication.
- uo
Perform potentially unsafe optimizations.
- ur
Perform loop unrolling.

Code-generation options

- c
Suppress loading.
- except precise
Ensures that any arithmetic exceptions issued by a subprogram before it returns will be received by the subprogram. Works only for compilation-time target machine. Degrades performance.
- except default
Cancels -except precise.
- fi
Use IEEE floating-point format.
- fn
Use native floating-point format.
- in
Specify default length (*n*) for INTEGER and LOGICAL variables; *n* can be 2, 4, or 8. Obsolete; use -p8 or -pd8 instead.
- mi *n*
n is the expected memory interleave on target machine.
- p8
Specify default length of 8 for INTEGER, LOGICAL, and REAL variables; specify default length of 16 for DOUBLE PRECISION and COMPLEX variables.
- pd8
Specify default length of 8 for INTEGER, LOGICAL, REAL, and DOUBLE PRECISION variables; specify default length of 16 for COMPLEX variables.

- rn*
Specify default length (*n*) for REAL variables; *n* can be 4 or 8. Obsolete; use `-p8` or `-pd8`.
- re
Generate reentrant code.
- s
Generate symbolic assembly code.
- tm *target*
Generate code for target machine. *target* = c1, c2, c31, c32, c34 or c38.

Message and listing options

- errnames
Output message name when a compile-time message is issued. Refer to Appendix C of the *CONVEX FORTRAN User's Guide* for more information.
- LST
Produce listing of compiled source program.
- LSTI
Produce `-LST` listing with `INCLUDE` files.
- na
Suppress advisory diagnostics.
- nv
This option is no longer supported. Use `-or none` instead.
- nw
Suppress warning diagnostics.
- or *table*
Produce specified *table*. *table* can be loop, array, all, or none.
- pl *n*
Specifies a default maximum page length of *n* lines per page for the `-LST`, `-LSTI`, `-xr`, and `-xra` options. *n* defaults to 60.

-pw *n*
Specifies a desired header width of *n* columns per line for the **-LST**, **-LSTI**, **-xr**, and **-xra** options. Data that exceeds this width is wrapped to the next line. *n* defaults to 80.

-xr
Invokes the cross-referencer and generates a cross-reference report at compilation time.

-xra
Invokes cross-referencer and generates data file but does not generate report. The report can be generated by running **fcxref**.

-iw *n*
Specifies the desired identifier width. *n* defaults to 16. Must be used with **-xr** or **-xra** option.

-xrf *file*
Use *file* as the cross-referencer data file instead of **.fcxrefData**. Must be used with **-xr** or **-xra**.

-xrr *file*
Sends the cross-reference report to *file* rather than to **stdout**. Must be used with **-xr** or **-xra** option.

-xrg *n*
Produces composite/global common block reports after all routine reports. Must be used with **-xr** or **-xra** option. *n* = desired sum of the following:

- 1: Exhaustive members list, ordered by member offsets.
- 2: Exhaustive members list, ordered by member names.
- 4: Differing members report, ordered by member offsets.
- 8: Differing members report, ordered by member names (default).

-xro
Calls the old **fxref** cross-reference generator.

-xrl
Calls the **fxref** cross-reference generator and puts all objects (such as variables and arrays) into one table.

Debugging and profiling options

- a1
Treat noncharacter arrays with last dimension of 1 as assumed-size (last dimension of *).
- cs
Check that subscripts are within array bounds.
- cxdb
Generate debugging information for CXdb.
- db
Produce debugging information for csd.
- dc
Compile lines with D in column 1.
- metrics
Writes the CXmetrics data file *sourcefile.met*, where *sourcefile* is the name of your FORTRAN source file without the .f or .FOR extension.
- p
Produce code for prof profiler.
- pb
Produce code for bprof profiler.
- pg
Produce code for gprof profiler.
- pa
Produce code for CXpa profiler routine and loop levels.
- pab
Produce code for CXpa profiler block level.
- par
Generate routine-level instrumentation for profiling under CXpa.
- sc
Check syntax only.

Miscellaneous options

- ansic
Links /usr/lib/libc.a to your program.

- B*directory***
Find the substitute compiler in the named directory.
- fpp**
Use preprocessor as first step of compilation.
- link *arg***
Passes *arg* to the loader.
- nosc**
Disable short circuit evaluation of conditionals.
- o *name***
Specify name for executable module.
- pcc**
Links a pcc-compatible `libc.a` to your program.
- tl *n***
Set maximum compile time to *n* minutes.
- vn**
Identify compiler version.
- 72**
Process only first 72 characters of each program line.

error utility

`fc [options] source_file.f |& error`
inserts error messages as comments into *source_file.f*.

Directives

The general format of a compiler directive is as follows:

`C$DIR directive [, directive]`

A directive pertaining to a loop must immediately precede that loop.

BEGIN_TASKS

Identify a sequence of tasks for parallel execution.

```
BEGIN_TASKS
...
NEXT_TASK
...
NEXT_TASK
...
END_TASKS
```

A sequence of tasks begins with a `BEGIN_TASKS` directive and ends with an `END_TASKS` directive. A `NEXT_TASK` directive precedes each individual task.

DO_PRIVATE

Declares a list of variables and/or arrays private to the immediately following `DO` loop.

```
DO_PRIVATE (varlist)
```

where

varlist

is a list of variables or arrays, separated by commas, that are to be private to the immediately following loop.

FORCE_PARALLEL

Parallelize the loop that follows regardless of any apparent loop dependencies. Do not allow interchange of outer loops for vectorization.

```
FORCE_PARALLEL
```

FORCE_PARALLEL_EXT

Parallelize the loop that follows regardless of any apparent loop dependencies.

```
FORCE_PARALLEL_EXT
```

FORCE_VECTOR

Vectorize the loop that follows regardless of any apparent recurrences.

FORCE_VECTOR

MAX_TRIPS

Specify the maximum number of times to execute the loop that follows.

MAX_TRIPS (*n*)

where

n

is the maximum iteration (trip) count.

NO_PARALLEL

Do not parallelize the loop that follows.

NO_PARALLEL

NO_PEEEL

Prevent the compiler from applying loop boundary value peeling to the loop that immediately follows. Overrides boundary level peeling at all levels: default, -peel, and -peelall.

NO_PEEEL

NO_PROMOTE_TEST

Prevent the compiler from applying test promotion to the loop that immediately follows. Overrides test promotion at all levels: default, -ptst, and -ptstall.

NO_PROMOTE_TEST

NO_RECURRENCE

Disregard any apparent recurrences in the loop that follows.

NO_RECURRENCE

NO_SIDE_EFFECTS

The specified functions do not modify the value of a parameter or common variable, read or write, or call another routine.

NO_SIDE_EFFECTS [(*func* [,*func*]...)]

where

func
is a user-defined function.

If no argument is given, the directive applies to all functions that follow.

NO_VECTOR

Do not vectorize the loop that follows.

NO_VECTOR

PEEL

Allow the compiler to peel the loop immediately following the directive, expanding the code beyond the default conservative limit, but not without bound.

PEEL

PEEL_ALL

Allow the compiler to peel the loop immediately following the directive, expanding the code without bound.

PEEL_ALL

PREFER_PARALLEL

Parallelize the loop that follows, if possible. Do not allow interchange of outer loops for vectorization.

PREFER_PARALLEL

PREFER_PARALLEL_EXT

Parallelize the loop that follows, if possible.

PREFER_PARALLEL_EXT

PREFER_VECTOR

Vectorize the loop that follows, if possible, if there is a choice of loops in a nest to vectorize.

PREFER_VECTOR

PROMOTE_TEST

Allow the compiler to promote tests out of the loop immediately following the directive, replicating code beyond the default conservative limit, but not without bound.

PROMOTE_TEST

PROMOTE_TEST_ALL

Allow the compiler to promote tests out of the loop immediately following the directive, replicating code without bound.

PROMOTE_TEST_ALL

PSTRIP

Strip-mine the following parallel loop using the specified length.

PSTRIP (*integer_constant*)

where

integer_constant

is an integer that specifies the strip-mine length.

ROW_WISE

The designated array names have their dimensions reversed.

```
ROW_WISE (array [,array]... )
```

where

array

is the name of the array.

SCALAR

Prevent the loop that follows from being vectorized or parallelized.

```
SCALAR
```

SELECT

Generate multiple versions of a loop and the code necessary to select, at runtime, which version to execute.

```
SELECT (vtrip, ptrip, potrip)
```

where

vtrip

is the iteration (trip) count at which the compiler selects vector execution for the loop.

ptrip

is the trip count at which the compiler selects parallel execution for the loop.

potrip

is the trip count at which the compiler selects parallel-vector execution for the loop.

SYNCH PARALLEL

Execute the loop that follows in parallel even though it requires synchronization that may decrease efficiency.

`SYNCH PARALLEL`

TASK PRIVATE

Declares a list of variables and/or arrays private to the immediately following task. Task is defined by `BEGIN_TASK`, `NEXT_TASK`, and `END_TASK` directives.

`TASK PRIVATE` (*varlist*)

where

varlist

is a list of variables or arrays, separated by commas, that are to be private to the immediately following task.

UNROLL

Perform loop unrolling on the body of the loop that follows.

`UNROLL`

VSTRIP

Strip-mine the following vector loop using the specified length.

`VSTRIP` (*k*)

where

k

is an integer constant.

Statements

ACCEPT

Sequentially read data from the implicit input unit.

```
ACCEPT f [, iolist]
```

or

```
ACCEPT * [, iolist]
```

or

```
ACCEPT namelist
```

where

f
is an expression or a format label.

*
specifies list-directed formatting.

iolist
is an I/O list.

namelist
is a namelist block name.

ALLOCATABLE

Declare an allocatable array.

```
[type arrexpr]
```

```
ALLOCATABLE (arrexpr [, ...])
```

where

type
is an optional type definition for the array. The form is identical to the form for a standard array type definition, except instead of specifying constant dimension parameters, you must either supply a rank definition or provide no parameters.

arrexp

is the array name or rank definition if it was not given in a preceding type statement. The array name may occur in both the type and ALLOCATABLE statements; the rank definition must occur in exactly one, or the compiler flags an error.

ALLOCATE

Allocate storage for an ALLOCATABLE array.

```
ALLOCATE (arrdef [, ...])
```

where

arrdef

is an array definition with subscripts or subscript ranges for all dimensions.

ASSIGN

Assign a statement label to an integer variable.

```
ASSIGN label TO i
```

where

label

is the label of an executable statement.

i

is an integer variable name.

BACKSPACE

Position a file connected to the specified unit to the beginning of the previous record.

```
BACKSPACE u
```

or

```
BACKSPACE ([UNIT=u] [, IOSTAT=ios] [, ERR=label])
```

where

u
is a logical unit specifier.

label
is the label of an executable statement.

ios
is an integer variable or integer array element.

BLOCK DATA

Define a subprogram that provides initial values for variables and array elements in common blocks.

```
BLOCK DATA [name]
```

where

name
is a symbolic name for the block data subprogram.

A block data subprogram begins with a BLOCK DATA statement and ends with an END statement.

CALL

Invoke a subroutine.

```
CALL subroutine [ ( [a [,a]... ] ) ]
```

where

subroutine
is the name of the subroutine.

a
is an actual argument.

CLOSE.

Disconnect a file from a unit.

```
CLOSE ( [UNIT=u] [, STATUS=p] [, ERR=lbl] [, IOSTAT=ios] )
```

or

```
CLOSE ([UNIT=u] [, DISP[OSE]=p]  
^      [, ERR=lbl] [, IOSTAT=ios] )
```

where

u
is a logical unit number.

p
is a character expression that determines the disposition of the file. Valid values are 'KEEP', 'SAVE', or 'DELETE'.

lbl
is the label of an executable statement.

ios
is an integer variable or integer array element.

COMMON

Allow variables or arrays in a main program or subprogram to share the same storage as variables and arrays in other subprograms.

```
COMMON [/cbn]/] nlist [[,]/cbn]/nlist]...
```

where

cbn
is a common block name.

nlist
is a list of variable names, array names, or array declarators.

CONTINUE

This executable statement has no effect.

```
CONTINUE
```

DATA

Establish initial values for arrays, array elements, substrings, and variables.

```
DATA nlist/clist / [ [, ] nlist/clist / ] ...
```

where

nlist

is a list of arrays, array elements, character substrings, implied DO lists, or variable names.

clist

is a list of constants or symbolic names of constants (defined with a PARAMETER statement).

DEALLOCATE

Deallocate an ALLOCATABLE array's storage.

```
DEALLOCATE (arrname [, ...])
```

where

arrname

is the name of the array to be deallocated. *arrname* must be free of subscripts.

DECODE

Transfer data between arrays or variables in internal storage, translating from character to internal form.

```
DECODE (c, f, b [, IOSTAT=ios] [, ERR=label]) [iolist]
```

where

c

is an integer expression.

f

is a format identifier.

b

is an array, array element, variable, or character substring reference that contains the characters to be translated.

ios

is either an integer array element or an integer variable that is defined as a positive integer if an error occurs or as zero if no error occurs.

label

is the label of an executable statement.

iolist

is an I/O list that receives the data.

DO

Begin a loop.

```
DO [ label [, ] ] var = expr1, expr2, expr3
```

where

label

is the label of an executable statement. If no label is specified, the loop must end with an END DO statement.

var

is the name of an integer, real, or double-precision variable.

expr1

is an integer, real, or double-precision expression that specifies the value of *var* on initial execution of the loop.

expr2

is an integer, real, or double-precision expression that specifies the ending value of *var*.

expr3

is an integer, real, or double-precision expression that specifies the increment for *var* after each iteration of the loop.

A DO loop begins with a DO statement and ends with an END DO statement or a statement marked by *label*.

DO WHILE

Execute a loop as long as a logical expression remains true.

```
DO [label [, ] ] WHILE (expr)
```

where

label

is the label of an executable statement.

expr

is a logical expression.

A DO WHILE loop begins with a DO WHILE statement and ends with an END DO or a statement marked by *label*.

DIMENSION

Specify the bounds of arrays.

```
DIMENSION array (dim [, dim] ... ) ...
```

where

array

is the name of an array.

dim

is a dimension of the array in the form

```
[[lower_bound]:] upper_bound.
```

ENCODE

Transfer data between arrays or variables in internal storage, translating from internal to character form.

```
ENCODE (c, f, b [, IOSTAT=ios] [, ERR=label] ) [iolist]
```

where

c

is an integer expression.

f

is a format identifier.

b

is an array, array element, variable, or character substring to receive the data.

ios

is either an integer array element or an integer variable that is defined as a positive integer if an error occurs or as zero if no error occurs.

label

is the label of an executable statement to which control transfers if an error occurs.

iolist

is an I/O list that contains the data to be translated.

END

Indicate the end of a program unit. END terminates a main program without a message or returns control from a function or subroutine.

END

ENDFILE

Write an endfile record on the file connected to a unit.

```
ENDFILE ([UNIT=]u [, IOSTAT=ios] [, ERR=label])
```

where

u

is a logical unit specifier.

label

is the label of an executable statement.

ios

is an integer variable or integer array element.

ENTRY

Specify an alternate entry point for a function or subroutine.

```
ENTRY name [ ( [d [,d]... ] ) ]
```

where

name

is the symbolic name of the entry point.

d

is a dummy argument.

EQUIVALENCE

Cause two or more entities in a program to refer to the same storage area.

```
EQUIVALENCE (list) [, (list) ]...
```

where

list

is a list of two or more variables, array elements, array names, or character substring names separated by commas.

EXTERNAL

Identify a name as representing an externally defined procedure or dummy procedure.

```
EXTERNAL n [, n]...
```

where

n

is the symbolic name of a subprogram, block data subprogram, or dummy procedure.

FIND

Position a direct-access file to a particular record and set the associated variable to that record number.

```
FIND ( [UNIT=n], REC=rec, [, ERR=label] [, IOSTAT=ios] )
```

where

n

is a logical unit number that refers to a direct-access file.

rec

is the direct-access record number.

label

is the label of an executable statement to which control transfers if an error occurs.

ios

is an integer variable or an integer array element that is defined as a positive integer if an error occurs or as zero if no error occurs.

FORMAT

Provide information to create the desired format for I/O statements.

label FORMAT (*list*)

where

label

is a required statement label.

list

is a nonempty format list.

FUNCTION

Specify the name of a function and its arguments.

[*type*] FUNCTION *name* [**m*] [([*d* [,*d*]...])]

where

type

is a logical, character, or numeric data type specifier.

name

is the name of the function subprogram.

m

is an unsigned, nonzero integer constant specifying the length of the data type.

d

is a dummy argument name.

A function subprogram begins with a FUNCTION statement and ends with an END statement.

GOTO

Transfer control to a designated statement.

GOTO *label*

or

GOTO (*list*) [,] *expr*

or

GOTO *var* [[,] (*list*)]

where

label

is the label of an executable statement.

list

is a list of labels of executable statements.

expr

is an arithmetic expression.

var

is an integer variable name defined by an ASSIGN statement with the value of an executable statement label.

IF

Transfer control based on the value of an arithmetic or logical expression.

IF (*arith-expr*) *label-1*, *label-2*, *label-3*

or

IF (*logical-expr*) *statement*

where

arith-expr

is an arithmetic expression of type integer or real.

label

is the label of an executable statement. For an arithmetic IF, all three labels must be given.

Control is transferred to *label-1* if the value of the expression is less than zero, *label-2* if it equals zero, and *label-3* if it is greater than zero.

logical-expr

is a logical expression.

statement

is an executable statement other than a DO, END DO, END, logical IF, or block IF.

IF THEN

Execute one or more statements depending on the value of a logical expression.

```
IF (logical-expr) THEN
  ...
END IF
```

or

```
IF (logical-expr) THEN
  ...
ELSE
  ...
END IF
```

or

```
IF (logical-expr-1) THEN
  ...
ELSE IF (logical-expr-2) THEN
  ...
ELSE IF (logical-expr-3) THEN
  ...
ELSE
  ...
END IF
```

where

logical-expr

is a logical expression.

Any number of ELSE IF . . . THEN statements may be specified. Use of the ELSE statement is optional.

IMPLICIT

Override the implied data typing of symbolic names.

```
IMPLICIT typ (a [,a] . . .) . . .
```

or

```
IMPLICIT NONE
```

where

typ

is an INTEGER[**len*], REAL[**len*], DOUBLE PRECISION, CHARACTER[**len*] DOUBLE COMPLEX, COMPLEX[**len*], LOGICAL[**len*], or STRUCTURE data type.

a

is a letter or range of letters.

len

is an optional length specifier.

The IMPLICIT NONE form overrides all implicit defaults.

INCLUDE

Include source code from the specified file into the program being compiled.

```
INCLUDE 'filename'
```

where

filename

is the path name of the file from which source code is to be read.

INQUIRE

Determine specific information about a file or unit.

```
INQUIRE (FILE=fi, list) or INQUIRE ([UNIT=u], list)
```

where

f_i

is a character expression, numeric array name, numeric variable name, or numeric array element name whose value is the name of the file being queried.

list

is a list of specifiers that indicate the information to be determined (see the *CONVEX FORTRAN Language Reference Manual*, Chapter 7).

u

is the external unit number.

INTRINSIC

Allow the use of intrinsic functions names as arguments to subprograms.

```
INTRINSIC n [,n]...
```

where

n

is one of the FORTRAN intrinsic functions.

NAMELIST

Associate a name with a list of variables or array names.

```
NAMELIST /name/varlist [ [, ] /name/varlist]...
```

where

name

is a symbolic name representing the list of entries to be read or written.

varlist

is a list of variable or array names, separated by commas, to be associated with *name*.

OPEN

Connect an existing external file to a unit, change the attributes of a connected file, or create a new file and connect it to a unit.

```
OPEN (keyword [,keyword] )
```

where

keyword

is a keyword and, possibly, an associated value (see the *CONVEX FORTRAN Language Reference Manual*, Chapter 7).

OPTIONS

Specify or override compiler options. If used, must be the first statement in the program.

```
OPTIONS option [option] ...
```

where

option

is a compiler option (see the *CONVEX FORTRAN Language Reference Manual*, Chapter 1).

PARAMETER

Assign a symbolic name to a constant.

```
PARAMETER (name=c [,name=c] ...)
```

or

```
PARAMETER name=c [,name=c] ...
```

where

name

is a symbolic name.

c

is a constant or constant expression.

The second form of the statement with just one constant is only available under the `-vfc` compiler option.

PAUSE

Temporarily halt program execution; resume execution on operator command.

```
PAUSE [message]
```

where

message

is the information to be displayed on halting and can be a character constant or a string of up to five digits.

PRINT

Transfer formatted records to the implicit output device using sequential access.

```
PRINT format [, iolist]
```

or

```
PRINT * [, iolist]
```

or

```
PRINT namelist
```

where

format

is a format specifier.

iolist

is an I/O list.

*

indicates list-directed formatting.

namelist

is a namelist specifier.

PROGRAM

Assign a name to the main program unit. This statement is optional.

```
PROGRAM name
```

where

name

is the symbolic name for the main program.

READ

Transfer data from an external file into internal storage, or transfer data between internal storage locations.

```
READ fmt [, iolist]
```

or

```
READ (u [, fmt] [, IOSTAT=ios, ERR=label, END=label]) [iolist]
```

or

```
READ (u, rec [, fmt]  
^      [, IOSTAT=ios, ERR=label, END=label]) [iolist]
```

or

```
READ (iu, fmt [, IOSTAT=ios, ERR=label, END=label]) [iolist]
```

where

fmt

is a format specifier or an asterisk (*) indicating list-directed format.

iolist

is an I/O list that identifies the data to be transferred.

u

is a logical unit number.

ios

is an integer variable or array element.

label

is the label of an executable statement.

rec

is a record number. The apostrophe before the record number is available only in -vfc mode.

iu

is the name of a character variable, array, array element, or substring used as an internal unit.

RETURN

Return control from a subroutine subprogram to the calling program.

```
RETURN [expr]
```

where

expr

is an optional integer expression that specifies an alternate return.

REWIND

Position a file to its initial point.

```
REWIND ([UNIT=]u [, IOSTAT=ios] [, ERR=label])
```

where

u

is a logical unit specifier.

label

is the label of an executable statement.

ios

is an integer variable or integer array element.

SAVE

Retain the values of designated variables and arrays in a subroutine or function when a RETURN or END statement is executed.

```
SAVE [name [, name] ...]
```

where

name

is a variable name, array name, or named COMMON block contained between slashes.

If the SAVE statement has no arguments, the values of all allowable entities are retained.

STOP

Terminate program execution.

```
STOP [string]
```

where

string

is a string of up to five digits or a character constant to be displayed.

SUBROUTINE

Specify the name of a subroutine and its arguments.

```
SUBROUTINE name [ ( [d [,d]... ] ) ]
```

where

name

is a symbolic name of the subroutine.

d

is a dummy argument list containing variable, dummy procedure, and array names, or asterisks (*).

A subroutine begins with a SUBROUTINE statement and ends with an END statement.

TYPE

Transfer formatted records to the implicit output device using sequential access.

```
TYPE format [, iolist]
```

or

```
TYPE * [, iolist]
```

or

```
TYPE namelist
```

where

format
is a format specifier.

iolist
is an I/O list.

*

indicates list-directed formatting.

namelist
is a namelist specifier.

WRITE

Transfer data from internal storage to an external unit, or transfer data between internal storage locations.

```
WRITE fmt [, iolist]
```

OR

```
WRITE (u [, fmt] [, IOSTAT=ios, ERR=label]) [iolist]
```

OR

```
WRITE (u, rec [, fmt]  
^      [, IOSTAT=ios, ERR=label]) [iolist]
```

OR

```
WRITE (iu, fmt [, IOSTAT=ios, ERR=label]) [iolist]
```

where

fmt
is a format specifier or an asterisk (*) indicating list-directed formatting.

iolist
is an I/O list that identifies the data to be transferred.

u
is a logical unit number.

ios
is an integer variable or array element.

label

is the label of an executable statement.

rec

is a record number. The apostrophe before the record number is available only in `-vfc` mode.

iu

is the name of a character variable, array, array element, or substring used as an internal unit.

CONVEX Computer Corporation
Richardson, Texas USA

Document No. 720-002430-003