FORTRAN/9000 Quick Reference

FORTRAN Statements

Program Unit Statements

`BLOCK DATA` [*name*]

Identifies the program unit as a block data subprogram.

`END`

Specifies the end of a main program, subroutine, function, or block data subprogram.

`ENTRY` *name*[([*arg1* [,*arg2*] ...])]

Provides an alternative entry into a function or subroutine.

[*type*] `FUNCTION` *name*[**len*] ([*arg1*[,*arg2*] ...])

Identifies the program unit as a function subprogram.

`PROGRAM` *name* [(*list*)]

Identifies the program unit as a main program.

`SUBROUTINE` *name* [([*arg1* [,*arg2*] ...])]

Identifies the program unit as a subroutine subprogram.

Specification Statements

`COMMON` [/[*blkname1*]/] *list1* [[,] / [*blkname2*]/ *list2*] ...

Reserves a block of memory that can be used by more than one program unit.

`DIMENSION` *name1*(*bounds*) [,*name2*(*bounds2*)] ...

Defines the dimensions and bounds of an array.

`EQUIVALENCE` (*list1*) [,(*list2*)] ...

Associates variables so that they share the same place in memory.

`EXTERNAL` *proc1* [,*proc2*] ...

Identifies subprogram names used as actual arguments or as non-intrinsics.

`IMPLICIT` *type*(*range1*[, *range2*]...)[,*type*(*range1* [, *range2*]...)]...

Specifies the type associated with the first letter of a symbolic name.

`IMPLICIT NONE`

Disables implicit typing.

`INCLUDE` *name*

Includes and processes source statements from a specified file or device.

`INTRINSIC` *fun*[,*fun2*] ...

Identifies intrinsic function names used as actual arguments.

name ([*parm1* [, *parm2*] ...]) = *expr*

Defines a one-statement function.

```
MAP
 field_def
 ⋮
END MAP
```

Defines a map block within a union in a structure.

`NAMELIST` /*groupname*/*varlist* [[,]/*groupname2*/*varlist2*] ...

Defines a namelist group for namelist-directed I/O.

`PARAMETER` (*cname1*=*cexp* [,*cname2*=*cexp*] ...)

Defines named constants.

`POINTER` (*p1*,*v1*) [,(*p2*,*v2*) ...]
`POINTER` /*p*/ *v1* [,*v2* ...]

Allows you to manage dynamic structures such as lists and variable-length arrays.

`RECORD` /*struc_name*/ *recnam*[,*recnam*] ... [,/*struc_name*/ *rec_list*] ...

Declares a record of a structure defined in a `STRUCTURE` statement.

`RETURN` [*rtnum*]

Transfers control from a subprogram back to the calling program unit.

`SAVE` [*var1* [,*var2*] ...]

Retains the value of an entity after execution of a `RETURN` or `END` statement in a subprogram.

```
STRUCTURE /name/
 field_def
 ⋮
END STRUCTURE
```

Defines a structure to be used in a `RECORD` declaration.

`UNION`

Defines a union inside a `STRUCTURE`.

`VOLATILE` *list*

Prevents optimization operations on specified entities.

Value Assignment Statements

`ASSIGN` *label* `TO` *variable*

Assigns a value to a variable used in a `GOTO` or as a `FORMAT` statement label.

var = *expr*

Assigns arithmetic, logical, character, or composite values to variables at execution time.

`DATA` *varlist1*/*conlist1*/[[,]*varlist2*/*conlist2*/] ...

Assigns values to variables before execution.

type name1[/*conlist*/] [,*name2*[/*conlist*/]] ...

Can optionally assign initial values to variables, in addition to its primary function as a type specification statement.

Control Statements

`CALL` *name*[([*arg1* [,*arg2*] ...])]

Transfers control to an external procedure.

`RETURN` [*rtnnum*]

Transfers control from a subprogram back to the calling program.

`CONTINUE`

Causes execution to continue—has no effect of its own.

`DO` [*label* [,]]*index*=*init*, *limit* [,*step*]

Labelled `DO` which causes the statement(s) up to and including the specified *label* to be executed a specified number of times.

```
DO index=init, limit [,step]
 statements
END DO
```

Block `DO` which causes a group of statements to be executed a specified number of times.

```
DO [label [,]] WHILE (lexpr)
 statements
END DO
```

Causes a group of statements to be executed until a specified condition is met.

`GOTO` *label*

Unconditional `GOTO` that transfers control to a specified statement.

`GOTO` (*label1*, *label2*, ...)[,] *exp*

Computed `GOTO` that transfers control to *label1* if *exp* is 1, *label2* if *exp* is 2, and so on.

`GOTO` *ivar* [[,] (*label1* [,*label2*] ...)]

Assigned `GOTO` that transfers control to the statement whose label was previously assigned to *ivar* by an `ASSIGN` statement.

`IF` (*exp*) *labeln*, *labelz*, *labelp*

Arithmetic `IF` that transfers control to *labeln* if *exp* is negative, *labelz* if *exp* is 0, and *labelp* if *exp* is positive.

`IF` (*exp*) *statement*

Logical `IF` that conditionally executes a statement based on a logical value.

```
IF (exp1) THEN
 statements
[ ELSE IF (exp2) THEN
 statements] ...
[ ELSE
 statements]
ENDIF
```

Block `IF` that executes optional groups of statements based on one or more conditions.

`ON` *interrupt_condition* { `CALL` *trap_routine* | `ABORT` | `IGNORE` }

Specifies the action to be taken after the subsequent interruption of a program's execution.

Input/Output Statements

`ACCEPT`

Equivalent to `READ(UNIT=5)`.

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

Positions a file at the previous record.

`CLOSE` ([`UNIT=`]*unit* [,`IOSTAT=`*ios*] [,`ERR=`*label*] [,`STATUS=`*stat*])

Terminates access to a file.

`DECODE` (*count*,[`FMT=`]*fmt*,[`UNIT=`]*unit* [,`IOSTAT=`*ios*][,`ERR=`*label*])[*list*]

Transfers data to an internal file.

`ENCODE` (*count*,[`FMT=`]*fmt*,[`UNIT=`]*unit* [,`IOSTAT=`*ios*][,`ERR=`*label*]) [*list*]

Transfers data from an internal file.

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

Writes an end-of-file.

label `FORMAT` (*des1* [,*des2*] ...)

Describes how input/output information is arranged.

`INQUIRE` ([`UNIT=`]*unit*, `FILE=`*name* [,`IOSTAT=`*ios*][,`ERR=`*label*] [*specifier_list*])

Supplies information about files.

`OPEN` ([`UNIT=`]*unit*[,`FILE =` *name*] [,`IOSTAT =` *ios*][,`ERR =` label] ...)

Allows access to a file.

`PRINT` *fmt* [,*list*]

Transfers data out in formatted form.

`PRINT *` [,*list*]

Transfers data out in list-directed form.

`PRINT` *nml*

Transfers data out in namelist-directed form.

`READ` ([`UNIT=`]*u*,[`FMT=`]*fmt*[,`END=`*end*] [,`ERR=`*err*][,`IOSTAT=`*ios*]) [*list*]
`READ` *fmt* [,*list*]

Formatted sequential `READ` statement.

`READ` ([`UNIT=`]*u*, * [,`END=`*end*] [,`ERR=`*err*][,`IOSTAT=`*ios*]) [*list*]
`READ *` [,*list*]

List-directed sequential `READ` statement.

`READ` ([`UNIT=`]*u*, [`NML=`]*nml* [,`END=`*end*][,`ERR=`*err*][,`IOSTAT=`*ios*])
`READ` *nml*

Namelist-directed sequential `READ` statement.

`READ` ([`UNIT=`]*u* [,`END=`*end*] [,`ERR=`*err*][,`IOSTAT=`*ios*]) [*list*]

Unformatted sequential `READ` statement.

`READ` ([`UNIT=`]*u*,[`FMT=`]*fmt*[,`REC=`*rn*] [,`ERR=`*err*][,`IOSTAT=`*ios*]) [*list*]

Formatted direct-access `READ` statement.

`READ` ([`UNIT=`]*u* [,`REC=`*rn*][,`ERR=`*err*] [,`IOSTAT=`*ios*]) [*list*]

Unformatted direct-access `READ` statement.

`READ` ([`UNIT=`]*iu*,[`FMT=`]*fmt*[,`END=`*end*] [,`ERR=`*err*][,`IOSTAT=`*ios*]) [*list*]

Formatted internal `READ` statement.

`READ` ([`UNIT=`]*iu*, * [,`END=`*end*][,`ERR=`*err*] [,`IOSTAT=`*ios*]) [*list*]

List-directed internal `READ` statement.

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

Positions a file at beginning-of-file.

`TYPE`

See `PRINT`.

`WRITE` ([`UNIT=`]*u*,[`FMT=`]*fmt*[,`ERR=`*err*] [,`IOSTAT=`*ios*])[*list*]

Formatted sequential `WRITE` statement

`WRITE` ([`UNIT=`]*u*, * [,`ERR=`*err*] [,`IOSTAT=`*ios*]) [*list*]

List-directed sequential `WRITE` statement

`WRITE` ([`UNIT=`]*u*, [`NML=`]*nml* [,`ERR=`*err*] [,`IOSTAT=`*ios*])

Namelist-directed sequential `WRITE` statement

`WRITE` ([`UNIT=`]*u*, [`ERR=`*err*] [,`IOSTAT=`*ios*]) [*list*]

Unformatted sequential `WRITE` statement

`WRITE` ([`UNIT=`]*u*,[`FMT=`]*fmt*[,`REC=`*rn*] [,`ERR=`*err*][,`IOSTAT=`*ios*])[*list*]

Formatted direct-access `WRITE` statement

`WRITE` ([`UNIT=`]*u* [,`REC=`*rn*] [,`ERR=`*err*] [,`IOSTAT=`*ios*]) [*list*]

Unformatted direct-access `WRITE` statement

`WRITE` ([`UNIT=`]*iu* ,[`FMT=`]*fmt* [,`ERR=`*err*] [,`IOSTAT=`*ios*]) [*list*]

Internal `WRITE` statement

`WRITE` ([`UNIT=`]*iu*, * [,`ERR=`*err*] [,`IOSTAT=`*ios*]) [*list*]

List-directed internal `WRITE` statement

Program Halt Statements

`PAUSE` [*n*]

Causes program suspension.

`STOP` [*n*]

Terminates program execution.

Declaration Statements

Type:Storage Declaration(s)	Range of Values

Integer: 4 bytes
`INTEGER` *v1*`[,`*v2* ... `]`
`INTEGER*4` *v1*`[,`*v2* ... `]`
	−2 147 483 648 to +2 137 483 647

Short Integer: 2 bytes `INTEGER*2` *v1*`[,`*v2* ... `]`	−32 768 to +32 767

Real: 4 bytes
`REAL` *v1*`[,`*v2* ... `]`
`REAL*4` *v1*`[,`*v2* ... `]`
	$-3.402\,823\times10^{+38}$ to
	$-1.175\,495\times10^{-38}$
	0
	$+1.175\,495\times10^{-38}$ to
	$+3.402\,823\times10^{+38}$

Double Precision: 8 bytes
`DOUBLE PRECISION` *v1*`[,`*v2* ... `]`
`REAL*8` *v1*`[,`*v2* ... `]`
	$-1.797\,693\,134\,862\,31\times10^{+308}$ to
	$-2.225\,073\,858\,507\,21\times10^{-308}$
	0
	$+2.225\,073\,858\,507\,21\times10^{-308}$ to
	$+1.797\,693\,134\,862\,31\times10^{+308}$

Quad Precision: 16 bytes `REAL*16` *v1*`[,`*v2* ... `]`	0.0 and $\pm3.362103143112093506262677817321753\times10^{-4932}$ to $\pm1.189731495357231765085759326628007\times10^{+4932}$

Complex: 8 bytes
`COMPLEX` *v1*`[,`*v2* ... `]`
`COMPLEX*8` *v1*`[,`*v2* ... `]`
	same as *real*

Double Complex: 16 bytes
`DOUBLE COMPLEX` *v1*`[,`*v2* ... `]`
`COMPLEX*16` *v1*`[,`*v2* ... `]`
	same as *double precision*

Logical: 4 bytes
`LOGICAL` *v1*`[,`*v2* ... `]`
`LOGICAL*4` *v1*`[,`*v2* ... `]`
	`.TRUE.` or `.FALSE.`
	same as *integer*

Short Logical: 2 bytes `LOGICAL*2` *v1*`[,`*v2* ... `]`	`.TRUE.` or `.FALSE.` same as *short integer*

Byte: 1 byte
`BYTE` *v1*`[,`*v2* ... `]`
`LOGICAL*1` *v1*`[,`*v2* ... `]`
	`.TRUE.` or `.FALSE.`
	−128 to +127
	entire 8-bit ASCII character set

Character: *len*
`CHARACTER*`*len* *v1*`[,`*v2* ... `]`
`CHARACTER*(*)` *v1*`[,`*v2* ... `]`
	entire 8-bit ASCII character set

HEWLETT PACKARD

Format Specifications

Edit Descriptors

Descriptor	Function
`BN`	Ignore blanks in numeric input data.
`BZ`	Treat blanks as zeros in numeric input data.
`NL`	End output with newline.
`NN`	Suppress newline at end of output.
`$`	Same as `NN`.
`Q`	Returns the number of bytes remaining to be read in the current input record.
`S`	Processor determines the sign output.
`SP`	Print optional plus signs on output.
`SS`	Inhibit optional plus signs on output.
`' ... '`	Single quotation marks for literal editing of output.
`" ... "`	Double quotation marks for literal editing of output.
n`H`	Hollerith editing for output only.
n`X`	Skip *n* columns to the right.
`T`*c*	Skip to column *c*.
`TL`*c*	Skip *c* columns to the left.
`TR`*c*	Skip *c* columns to the right.
`/`	Terminate current record and begin new record.
`:`	Terminate format control if remaining list is empty.
k`P`	Scale factor for input and output.

Format Descriptors

Descriptor	Explanation
Numeric:	
[*r*]`I`[*w*[.*d*]]	integer and short integer.
[*r*]`F`[*w*[.*d*]]	fixed-point format descriptor for real, double precision, `REAL*16`, complex, and double complex.
[*r*]`D`[*w*[.*d*]] [*r*]`E`[*w*[.*d*]][`E`*e*]	floating-point format descriptor for real, double precision, `REAL*16`, complex, and double complex.
[*r*]`G`[*w*[.*d*]][`E`*e*]	fixed or floating-point format for real, double precision, `REAL*16`, complex, and double complex descriptors.
Character:	
[*r*]`A`[*w*]	character data, left-justified in memory and external format.
[*r*]`R`[*w*]	character data, right-justified in memory and external format.
Logical:	
[*r*]`L`[*w*]	logical, short logical, and byte.
Octal and Hexadecimal:	
[*r*]`K`[*w*] [*r*]`O`[*w*] [*r*]`O`[*w*]	octal; integer and short integer.
[*r*]`Z`[*n*[.*m*]]	hexadecimal; any data type.

Compiler Directives

`ALIAS`	`IF`	`NOSTANDARD OPEN`
`ANSI`	`INCLUDE`	`NOSTANDARD SYSTEM`
`APOLLO LOGICALS`	`INIT`	`ONETRIP`
`APOLLO INTRINSICS`	`INLINE`[1]	`OPTIMIZE`
`ASSERT`[1]	`LINE`[1]	`OPTION`
`ASSERTIONS`[1]	`LINES`	`PAGE`
`CHECK_OVERFLOW`	`LIST`	`POSTPEND`
`CONTROL`	`LITERAL_ALIAS`	`RANGE`
`CROSSREF`	`LOGICAL OP`	`RENAME_COMMON`
`DEBUG`	`LOGICAL TEST`	`ROUNDOFF`[1]
`ELSE`	`LOGICAL TRUE`	`SAVE_LOCALS`
`ENDIF`	`LONG`	`SET`
`ERROR`[1]	`LOWERCASE`	`SHARED_COMMON`
`HP_DESTINATION ARCHITECTURE`	`MINVECTOR`[1]	`SHORT`
`HP_DESTINATION SCHEDULER`	`MIXED_FORMATS`	`SUBTITLE`
`HP1000 ALIGNMENT`	`MP ERROR`[1]	`SYMTABLE`
`HP1000 STRING_MOVE`	`MP INFO`[1]	`TABLES`
`HP9000_300`	`MP SUMMARY`[1]	`TITLE`
`HP9000_300 ALIGNMENT`	`MP WARNING`[1]	`UNROLL`[1]
`HP9000_300 LOGICALS`	`NLS`	`UPPERCASE`
`HP9000_800`	`NLS COMPARE`	`VECTORIZE`[1]
`HP9000_800 ALIGNMENT`	`NLS LITERALS`	`WARNING`[1]
`HP9000_800 LOGICALS`	`NOSTANDARD`	`WARNINGS`
`HP9000_800 INTRINSICS`	`NOSTANDARD LOGICALS`	`XREF`

[1]FTNOPP directives, described in Chapter 17, "Optimizing Preprocessor," of the *FORTRAN/9000 Reference*.

See Chapter 3, "Compiling FORTRAN Programs," of the *FORTRAN/9000 Reference*.

Default Tables

Default Compiler Table Sizes

Table Description	Default[1]
equivalence	150 EQUIVALENCE lists per program unit
statement label	201 labels per procedure, either user-specified or compiler-generated
external symbol	200 unique `COMMON`, `SUBROUTINE`, and/or `FUNCTION` names per file
control statements	maximum nesting of 20 `DO` and/or `IF` blocks
hash symbol	401 variable, array, `COMMON`, and procedure names per program
expression tree nodes	1000 nodes per statement in code generator
external symbol table storage	40000 bytes of `COMMON`, procedure, and static variable names per file—every reference is counted

[1]Table sizes are automatically expanded.

Miscellaneous Default Values

Description	Default
Length of a symbolic name	255 characters allowed; ANSI standard is 6
Number of continuation lines	99 allowed; ANSI standard is 19
Number of array dimensions	20 allowed; ANSI standard is 7

Intrinsics

ABS	BSIGN	DEXP	HMOD	IMINO	JNOT	QFLOTJ
ACOS	BTEST	DFLOAT	HMVBITS	IMIN1	JNUM	QINT
ACOSD	CABS	DFLOTI	HNOT	IMOD	JZEXT	QLOG
ACOSH	CCOS	DFLOTJ	HSHFT	INDEX	LEN	QLOG10
AIMAG	CDABS	DIM	HSHFTC	ININT	LGE	QMAX1
AIMAXO	CDCOS	DIMAG	HSIGN	INUM	LGT	QMIN1
AIMINO	CDEXP	DINT	HTEST	INOT	LLE	QMOD
AINT	CDLOG	DLOG	IABS	INT	LLT	QNINT
AJMAXO	CDSIN	DLOG10	IAND	IOR	LOG	QNUM
AJMINO	CDSQRT	DMAX1	IARGC	IQINT	LOG10	QPROD
ALOG	CEXP	DMIN1	IBCLR	IQNINT	MALLOC	QSIGN
ALOG10	CHAR	DMOD	IBITS	IRAND	MAX	QSIN
AMAXO	CLOG	DNINT	IBSET	IRANP	MAXO	QSIND
AMAX1	CMPLX	DNUM	ICHAR	ISHFT	MAX1	QSINH
AMINO	CONJG	DPROD	IDIM	ISHFTC	MIN	QSQRT
AMIN1	COS	DREAL	IDINT	ISIGN	MINO	QTAN
AMOD	COSD	DSIGN	IDNINT	IXOR	MIN1	QTAND
ANINT	COSH	DSIN	IEOR	IZEXT	MOD	QTANH
ASIN	CSIN	DSIND	IFIX	JIABS	MVBITS	RAND
ASIND	CSQRT	DSINH	IGETARG	JIAND	NINT	REAL
ASINH	CTAN	DSQRT	IIABS	JIBCLR	NOT	RNUM
ATAN	DABS	DTAN	IIAND	JIBITS	QABS	SIGN
ATAN2	DACOS	DTAND	IIBCLR	JIBSET	QACOS	SIN
ATAN2D	DACOSD	DTANH	IIBITS	JIDIM	QACOSD	SIND
ATAND	DACOSH	EXP	IIBSET	JIDINT	QACOSH	SINH
ATANH	DASIN	FLOAT	IIDIM	JIDNNT	QASIN	SIZEOF
BABS	DASIND	FLOATI	IIDINT	JIEOR	QASIND	SNGL
BADDRESS	DASINH	FLOATJ	IIDNNT	JIFIX	QASINH	SNGLQ
BBCLR	DATAN	FNUM	IIEOR	JINT	QATAN	SRAND
BBITS	DATAN2	FREE	IIFIX	JIOR	QATAND	SQRT
BBSET	DATAN2D	FSET	IINT	JIQINT	QATANH	TAN
BBTEST	DATAND	FSTREAM	IIOR	JIQNNT	QATAN2	TAND
BDIM	DATANH	GETARG	IIQINT	JISHFT	QATAN2D	TANH
BIAND	DBLE	GRAN	IIQNINT	JISHFTC	QCOS	ZABS
BIOR	DBLEQ	HABS	IISHFT	JISIGN	QCOSD	ZCOS
BITEST	DCMPLX	HBCLR	IISHFTC	JIXOR	QCOSH	ZEXP
BIXOR	DCONJG	HBITS	IISIGN	JMAXO	QDIM	ZLOG
BJTEST	DCOS	HBSET	IIXOR	JMAX1	QEXP	ZSIN
BMVBITS	DCOSD	HDIM	IJQNINT	JMINO	QEXT	ZSQRT
BMOD	DCOSH	HIAND	IMAG	JMIN1	QEXTD	ZTAN
BNOT	DDIM	HIEOR	IMAXO	JMOD	QFLOAT	
BSHFT	DDINT	HIXOR	IMAX1	JNINT	QFLOTI	
BSHFTC						

See Chapter 9, "Intrinsic Functions," of the *FORTRAN/9000 Reference*.

HP-UX ASCII Character Set

bits 3—0	bits 6—4							
	$^6000^4$	$^6001^4$	$^6010^4$	$^6011^4$	$^6100^4$	$^6101^4$	$^6110^4$	$^6111^4$
$^300000^0$	NUL	DLE	␣	0	@	P	‘	p
0001	SOH	DC1	!	1	A	Q	a	q
0010	STX	DC2	”	2	B	R	b	r
0011	ETX	DC3	#	3	C	S	c	s
0100	EOT	DC4	$	4	D	T	d	t
0101	ENQ	NAK	%	5	E	U	e	u
0110	ACK	SYN	&	6	F	V	f	v
0111	BEL	ETB	’	7	G	W	g	w
1000	BS	CAN	(	8	H	X	h	x
1001	HT	EM	)	9	I	Y	i	y
1010	LF	SUB	*	:	J	Z	j	z
1011	VT	ESC	+	;	K	[	k	{
1100	FF	FS	,	<	L	\	l	|
1101	CR	GS	-	=	M	]	m	}
1110	SO	RS	.	>	N	^	n	~
1111	SI	US	/	?	O	_	o	DEL

For example, for the "K" character bits 6 through 4 are 100 and bits 3 through 0 are 1011. Thus, the binary value of the "K" is 1001011, or 113 in octal.

FORTRAN Character Set

Char	Description
`A` to `Z`[1]	Uppercase Letters
a to z[1]	Lowercase Letters
`0` to `9`[1]	Digits
␣	Blank (Space)
$^H{}_T$	Tab
=	Equal Sign
+	Plus
−	Minus
*	Asterisk
/	Slash
(	Left Parenthesis
)	Right Parenthesis
,	Comma
.	Decimal Point

Char	Description
’	Single Quotation Mark (or Apostrophe)
:	Colon
$[1]	Dollar Sign
_[1]	Underscore
”	Double Quotation Mark
!	Exclamation Point
<	Left Angle Bracket
>	Right Angle Bracket
&	Ampersand
%	Percent

[1]These characters may appear within a symbolic name.