Fortran 95/2003 Quick Reference PDF (unofficial) 09-06-2007

Disclaimer: Ver 002.002

This information has not been approved by any formal draft body, therefore you use this information at your own risk; no one shall be held responsible for any consequences following the use of the information contained herein in any way whatsoever.

References Consulted include:
gfortran.pdf : Using GNU Fortran, For gcc version 4.2.1,(c) 2007 FSF
07-007r1.pdf: WORKING DRAFT, J3/07-007r1, 30th March 2007 9:52
n3661.pdf, ISO/IEC JTC1/SC22/WG5 N1578, ISO/IEC JTC1/SC22
N1579.pdf: ISO/IEC JTC1/SC22/WG5 N1579,New Features Fortran '03
2008 extensions: http://www.fortran.bcs.org/2007/bcs07.pdf

NOTE: This Quick Reference contains GNU-Extensions to the Fortran Standard, so please be aware of that while referring to this guide.

GNU Fortran 95/2003 Language Syntax and Quick Reference Guide
This guide favors the GNU gfortran series of compilers and utilities, each copyright their respective owners.

At the time of this writing, the following URL's provide downloads for the GNU Fortran packages:
http://gcc.gnu.org/wiki/GFortranBinaries
Win32 Version was known to be available here:
http://quatramaran.ens.fr/~coudert/gfortran/gfortran-windows.exe
Documentation can be downloaded here (for various GNU packages):
http://gcc.gnu.org/onlinedocs/

Finding out the version of gfortran installed (if any):

```
gfortran --version
GNU Fortran (GCC) 4.3.0 20070722 (experimental)
```

Most simple form of compile syntax: gfortran x.f95,
results in the creation of a.out, which is then executed: ./a.out
To formally name the executable application:
gfortran x.f95 -o x, execute the application: ./x

The following extensions are supported:

.f	Fortran 77	"generic fortran"	
.f95	Fortran95	.f90 Fortran 90	.f03 Fortran 2003

If you need to compile under a specific standard, use -std:
gfortran -std=f95, f2003, gnu, or legacy

\~~~~~~

Influencing runtime behavior with environment variables:
GFORTRAN_STDIN_UNIT—Unit number standard input
GFORTRAN_STDOUT_UNIT—Unit number standard output
GFORTRAN_STDERR_UNIT—Unit number standard error
GFORTRAN_USE_STDERR—Send lib output to standard error
GFORTRAN_TMPDIR—Directory for scratch files
GFORTRAN_UNBUFFERED_ALL—Don't buffer output
GFORTRAN_SHOW_LOCUS—Show location runtime errors
GFORTRAN_OPTIONAL_PLUS—Print leading + where permitted
GFORTRAN_DEFAULT_RECL—Default record length for new files
GFORTRAN_LIST_SEPARATOR—Separator list output
GFORTRAN_CONVERT_UNIT—Set endianness unformatted I/O

\~~~~~~

LANGUAGE-REFERENCE (WORK-IN-PROGRESS!)

Fortran character set consists of
- letters: ABCDEFGHIJKLMNOPQRSTUVWXYZ, abcdefghijklmnopqrstuvwxyz
- digits: 0123456789
- special characters: <blank> = + - * / ( ) , . ' : ! " % & ; < > ? $
- and underscore character '_'.

Special characters are used as operators, as separators or delimiters, or for grouping. '?' and '$' have no special meaning.
Lower case letters are equivalent to corresponding upper-case letters except in CHARACTER literals. Underscore character can be used as a non-leading significant character in a name.

\~~~~~~

Type	Kind Type Parameter	Notes
INTEGER	1	Range: -128 to 127
INTEGER	2	Range: -32,768 to 32,767
INTEGER	4*	Range: -2,147,483,648 to 2,147,483,647
INTEGER	8	Range: -9,223,372,036,854,775,808 to 9,223,372,036,854,775,807
REAL	4*	Range: 1.18 * 10E-38 to 3.40 * 10E38 Precision: 6-7 decimal digits
REAL	8	Range: 2.23 * 10E-308 to 1.79 * 10E308 Precision: 15-16 decimal digits
REAL	16	Range: 10E-4931 to 104932 Precision: approximately 33 decimal digits
COMPLEX	4*	Range: 1.18 * 10-38 to 3.40 * 1038 Precision: 7-8 decimal digits
COMPLEX	8	Range: 2.23 * 10E-308 to 1.79 * 10E308 Precision: 15-16 decimal digits
COMPLEX	16	Range: 10-4931 to 104932 Precision: approximately 33 decimal digits
LOGICAL	1	Values: .TRUE. and .FALSE.
LOGICAL	4*	Values: .TRUE. and .FALSE.
CHARACTER	1*	ASCII character set

\~~~~~~

Named Data: Implicit Typing, Type Declaration Statements, Attributes (DIMENSION, PARAMETER, POINTER, TARGET, EXTERNAL, ALLOCATABLE, {INTENT(IN), INTENT(OUT), INTENT(IN OUT) }, PUBLIC, PRIVATE, INTRINSIC, OPTIONAL, SAVE, SEQUENCE)

\~~~~~~

Substrings: string ([lower-bound] : [upper-bound])
Arrays, Array References, Array Elements,
Array Element Order:
$(1+(s1-j1))+((s2-j2)\times d1)+\ldots+((sn-jn)\times d\,n-1\times d\,n-2 \ldots\times d1)$ si is subscript in ith dimension, ji is lower bound of the ith dimension, di is size of the ith dimension, n is rank of the array,
Array Sections, Subscript Triplets, Vector Subscripts.

Arrays and Substrings: character (len=10), dimension (10,10) :: my_string
my_string(3:8,:) (2:4) = 'abc'

Dynamic Arrays.

Allocatable Arrays: integer, allocatable :: a(:), b(:,:,:)
allocate (a(3), b(1,3,-3:3))

Array Pointers:
```
integer, pointer, dimension(:,:) :: c
integer, target, dimension(2,4) :: d
integer, pointer, dimension(:,:) :: c
c => d
```

Assumed-Shape. Assumed-Size. Adjustable and Automatic Arrays.

Array Constructors: (/ constructor-values /)
```
    integer, dimension(3) :: a, b=(/1,2,3/), c=(/(i, i=4,6)/)
    a = b + c + (/7,8,9/) ! a is assigned (/12,15,18/)
   real,dimension(2,2) :: a = reshape((/1,2,3,4/),(/2,2/))
```

\~~~~~~

Operator	Represents	Operands
**	exponentiation	two numeric
* and /	multiplication and division	two numeric
+ and -	unary addition and subtraction	one numeric
+ and -	binary addition and subtraction	two numeric
//	concatenation	two CHARACTER
.EQ. and ==	equal to	2 numeric or 2 CHARACTER
.NE. and /=	not equal to	——————
.LT. and <	less than	
.LE. and <=	less than or equal to	two non-COMPLEX
.GT. and >	greater than	numeric or two CHARACTER
.GE. and >=	greater than or equal to	
.NOT.	logical negation	one LOGICAL
.AND.	logical conjunction	two LOGICAL
.OR.	logical inclusive disjunction	two LOGICAL
.EQV. and.NEQV.	logical equivalence and non-equivalence	

\~~~~~~

[GFORTRAN QUICK-REFERENCE GUIDE PAGE 2]
LANGUAGE-REFERENCE:

~~~~~~

Number-base BOZ Literal Constants (and GNU extensions):

```
Hexadecimal: Z’ABC’ and ’ABC’Z are equivalent
             data  z   /Z'688be87a'/
Binary:      B'01101' and '01101'B  are equivalent
             data  b  /B'01101000100010111110100001111010'/
Octal:       O'1504' and '1504'O are equivalent
             data  o  /O'15042764172'/
```

When converting from a LOGICAL to an INTEGER,
.FALSE. is interpreted as zero, and
.TRUE. is interpreted as one.
When converting from INTEGER to LOGICAL, the
value zero is interpreted as .FALSE. and
any nonzero value is interpreted as .TRUE..

RESULT = LOGICAL(L [, KIND]) Converts one kind of LOGICAL variable to another. Return value is a LOGICAL value equal to L, with a kind corresponding to KIND, or of the default logical kind if KIND is not given.

~~~~~~

```
BLOCK DATA [ block-data-name ]
 [ specification statement ] ...
END [ BLOCK DATA [ block-data-name ] ]
```

~~~~~~

CASE Construct

```
    [construct-name :] SELECT CASE (case-expr)
    CASE (case-selector [, case-selector] ... ) [construct-name]
          block
    [CASE DEFAULT [construct-name]]
          block
    END SELECT [construct-name]
```

~~~~~~

```
 COMMON [/ [common-name] /] common-object-list [[,]
                 / [common-name] / common-object-list] ...
```

statement provides a global data facility. It specifies contiguous blocks of physical storage, called common blocks, that are available to any program unit that references the common block.

~~~~~~

CONTAINS statement separates body of a main program, module, or subprogram from any internal or module subprograms it contains. See important notes in the Manual.

~~~~~~

Computed GOTO Statement (obsolescent)
GO TO (labels) [,] scalar-int-expr

~~~~~~

CYCLE statement skips to the next iteration of a DO loop.

```
    integer :: i, j
outer: do i=1, 10
     if(i < 3) cycle          ! cycles outer
inner:   do j=1, 10
       if (i < j) cycle       ! cycles inner
       if (i > j) cycle outer ! cycles to outer
     end do inner
    end do outer
```

~~~~~~

CONTINUE statement is traditionally used in conjunction with a statement label, as target of a branch statement or a do loop terminus. Execution of a CONTINUE statement has no effect; drops down to next statement.

~~~~~~

DATA statement provides initial values for data objects.

```
    DATA data-stmt-set [[,] data-stmt-set] ...
```

~~~~~~

DO construct specifies the repeated execution (loop) of a block of code.

```
[construct-name :] DO [label] [loop-control]
     block
    [exit]
do-termination
```

~~~~~~

Implied-DO loop allows elements to be transferred selectively or in some non-standard order. Rules for an implied-DO are similar to that of an ordinary DO-loop but loop forms a single item in the data-transfer list and is enclosed by a pair of parentheses.

~~~~~~

ENTRY entry-name [([dummy-arg-list]) [RESULT (result-name)]]
statement permits a program unit to define multiple procedures, each with a different entry point.

~~~~~~

EQUIVALENCE equivalence-sets
statement specifies two or more aliases that share same storage.

~~~~~~

EXTERNAL [::] external-name-list
statement declares external procedures. Specifying a procedure name as EXTERNAL permits the procedure name to be used as an actual argument.

~~~~~~

```
[construct-name:] FORALL ( forall-triplets [, mask] )
          [forall-body]
END FORALL [construct-name]
```

construct controls execution of a block of assignment and pointer assignment statements. Execution in block is selected by sets of index values and an optional mask expression.

~~~~~~

FORALL (forall-triplets [, mask]) forall-assignment-stmt
statement controls execution of an assignment or pointer assignment statement with selection by sets of index values and an optional mask expression.

~~~~~~

Format. Table Format Options. Single-line examples. [See Manual.]

~~~~~~

```
  function-name ( [dummy-args] ) = scalar-expr
   mean(a,b)=(a+b)/2
   c=mean(2.0,3.0) ! c is assigned value 2.5
```

Statement function is a function defined by a single statement.

~~~~~~

[PURE][ELEMENTAL][RECURSIVE] [type-spec] FUNCTION function-name ([dummy-arg-names]) [RESULT (result-name)]

~~~~~~

GO TO statement transfers control to a statement identified by a label.
GO TO label

~~~~~~

IF Construct

```
[construct-name:] IF (expr) THEN
        block
[ELSE IF (expr) THEN [construct-name]
        block]
[ELSE [construct-name]
        block]
END IF [construct-name]
```

~~~~~~

IF statement controls whether or not a statement is executed based on value of a logical expression. IF (expr) action-statement

~~~~~~

IMPLICIT implicit-specs, or IMPLICIT NONE
statement specifies a type and optionally a kind or a CHARACTER length for each variable or function name beginning with letter(s) specified in IMPLICIT statement. Alternately, it can specify that no implicit typing is to apply in the scoping unit.

~~~~~~

INCLUDE filespec
line causes text in a separate file to be processed as if text replaced INCLUDE line. INCLUDE line is not a Fortran statement.

~~~~~~

INTENT(IN, or OUT or IN OUT) [::] comma-separated dummy-args
statement specifies the treatment dummy arguments.

~~~~~~

```
 INTERFACE [generic-spec]
 [ procedure-heading
  [ specification-construct ] ...
```

```
LANGUAGE-REFERENCE:
  procedure-ending ] ...
   [ MODULE PROCEDURE module-procedure-name-list ] ...
  END INTERFACE [generic-spec]
   block specifies forms of reference by which a procedure can be
   invoked. An interface block specifies a procedure interface, a defined
  operation, or a defined assignment.
~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
INTRINSIC [::] intrinsic-procedure-names
  statement permits a reference to a specific intrinsic function as an actual
  argument.
~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
MODULE module-name
 [ specification construct ]
 [ CONTAINS
   subprogram [ subprogram ] ... ]
END [ MODULE [ module-name ] ]
statement begins a module program unit. Module encapsulates data and
procedures, provides a global data facility, which can be considered a
replacement for COMMON, and establishes implicit interfaces for
procedures contained in the module.
MODULE PROCEDURE module-procedure-list
statement can only appear in a generic interface block within a module or
within a program unit that accesses a module by use association.
~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
NULL function returns a disassociated pointer.
NULL ( [mold] )
real,pointer,dimension(:) :: a => null() ! a is disassociated
LANGUAGE-REFERENCE  continued (work-in-progress):
~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
NULLIFY statement disassociates a pointer.
NULLIFY (pointers)
~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
OPTIONAL [::] dummy-arg-names
statement declares that any dummy arguments specified need
not be associated with an actual argument when procedure is invoked.
~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
PARAMETER (named-constant-defs)
statement specifies and initializes named constants.
PAUSE (Obsolete)    Can be replaced by one WRITE and one READ
statement: more flexible/less system-dependent.
~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
pointer => target
Pointer assignment statement associates a pointer with a target.
~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
 POINTER [::] variable-name [(deferred-shape)]
                      [, variable-name [(deferred-shape)]] ...
statement specifies a list of variables that have POINTER attribute.
~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
PRIVATE [[::] access-ids]
statement specifies that names of entities are accessible only within
current module.
~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
PROGRAM program-name
 [ specification construct ]
 [ executable construct ]
[ CONTAINS
  internal-procedure [ internal-procedure ] ... ]
END [ PROGRAM [ program-name ]]
statement signals beginning of a main program unit.
~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
PUBLIC [[::] access-ids]
statement specifies that entities are accessible by use association anywhere
module that contains the PUBLIC statement is used.
~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
READ (io-control-specs) [inputs] or READ format [, inputs]
statement transfers values from an input/output unit to data objects
specified in an input list or a namelist group.
~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
REAL (a [, kind] )   function converts a number to a REAL data type.
REAL [kind-selector] [[, attribute-list] ::] entity [, entity] ...
           statement declares entities having REAL data type.
~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
RESULT  result_name
if specified, result_name becomes a function's result variable.
~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
RETURN [alt-return]
statement causes a transfer of control from a subprogram back to calling
procedure. Execution continues at statement following procedure
invocation.
~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
SAVE [[::] comma-separated list of object-name
                                        or / common-block-name /]
statement specifies that all data objects listed retain any previous
association, allocation, definition, or value upon reentry of a subprogram.
~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
SEQUENCE statement specifies a storage sequence for objects of a
   derived type. It can only appear within a derived type definition.
~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
STOP [scalar CHARACTER constant or a series of 1 to 5 digits]
statement causes execution of a program to terminate.
~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
TARGET [::] object-name [(array-spec)][,object-name [(array-spec)]] ...
statement specifies that data objects have target attribute and thus can be
associated with a pointer.
~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
  Definition:   TYPE [[, access-spec] ::] type-name
  Declaration: TYPE (type-name) [, attribute-list ::] entity [, entity] ...
statement defines a derived type, and declares entities having a derived
type.
~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
USE module [, rename-list]  or  USE module, ONLY: [only-list]
statement specifies that a module is accessible from current scoping unit.
It also provides a means of renaming or limiting the accessibility of
entities in the module.
~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
TYPE [ [, access-spec]::] type-name
 [ PRIVATE ]
 [ SEQUENCE ]
 [type-spec [[, component-attribute-list]::] &
   component-declaration-list ] ...
END TYPE [ type-name ]
~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
WHERE (LOGICAL mask-expr)
       [assignment-stmt]
[ELSEWHERE (LOGICAL mask-expr)]
       [assignment-stmt]
[ELSE WHERE]
       [assignment-stmt]
END WHERE
construct controls which elements of an array will be affected by a block
of assignment statements. Also known as masked array assignment.
~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
WRITE (io-control-specs) [outputs]
statement transfers values to an input/output unit from entities specified
in an output list or a namelist group.
[UNIT =] io-unit  or [FMT =] format  or [NML =] namelist-group-name
  or REC=record  or IOSTAT=stat  or ERR=errlabel  or END=endlabel
  or EOR=eorlabel  or ADVANCE=advance  or SIZE=size
io-unit is an external file unit, or *
format is a format specification
record is the number of the direct-access record that is to be written.
stat is a scalar default INTEGER variable that is assigned a positive value
if an error condition occurs and zero otherwise.
errlabel is a label that is branched to if an error condition occurs and no
end-of-record condition or end-of-file condition occurs during execution
of the statement.
endlabel is a label that is branched to if an end-of-file condition occurs and
no error condition occurs during execution of the statement.
eorlabel is a label that is branched to if an end-of-record condition occurs
and no error condition or end-of-file condition occurs during execution of
the statement.
advance is a scalar default CHARACTER expression that evaluates to NO
```

if non-advancing input/output is to occur, and YES if advancing input/output is to occur. The default value is YES.

size is a scalar default INTEGER variable that is assigned the number of characters transferred by data edit descriptors during execution of the current non-advancing input/output statement.

~~~~~~

13.5.1 Numeric functions

ABS (A)	Absolute value
AIMAG (Z)	Imaginary part of a complex number
AINT (A [, KIND])	Truncation to whole number
ANINT (A [, KIND])	Nearest whole number
CEILING (A [, KIND])	Least integer greater than or equal to number
CMPLX (X [, Y, KIND])	Conversion to complex type
CONJG (Z)	Conjugate of a complex number
DBLE (A)	Conversion to double precision real type
DIM (X, Y)	Positive difference
DPROD (X, Y)	Double precision real product
FLOOR (A [, KIND])	Greatest integer less than or equal to number
INT (A [, KIND])	Conversion to integer type
MAX (A1, A2 [, A3,...])	Maximum value

~~~~~~

13.5.1 Numeric functions

MIN (A1, A2 [, A3,...])	Minimum value
MOD (A, P)	Remainder function
MODULO (A, P)	Modulo function
NINT (A [, KIND])	Nearest integer
REAL (A [, KIND])	Conversion to real type
SIGN (A, B)	Transfer of sign

~~~~~~

13.5.2 Mathematical functions

ACOS (X)	Arccosine
ASIN (X)	Arcsine
ATAN (X)	Arctangent
ATAN2 (Y, X)	Arctangent
COS (X)	Cosine
COSH (X)	Hyperbolic cosine
EXP (X)	Exponential
LOG (X)	Natural logarithm
LOG10 (X)	Common logarithm (base 10)
SIN (X)	Sine
SINH (X)	Hyperbolic sine
SQRT (X)	Square root
TAN (X)	Tangent
TANH (X)	Hyperbolic tangent

~~~~~~

13.5.3 Character functions

ACHAR (I [, KIND])	Character in given position in ASCII collating sequence
ADJUSTL (STRING)	Adjust left
ADJUSTR (STRING)	Adjust right
CHAR (I [, KIND])	Character in given position in processor collating sequence
IACHAR (C [, KIND])	Position of a character in ASCII collating sequence
ICHAR (C [, KIND])	Position of a character in processor collating sequence
INDEX (STRING,SUBSTRING [,BACK,KIND])	Starting position of a substring
LEN TRIM (STRING [, KIND])	Length without trailing blank characters
LGE (STRING A, STRING B)	Lexically greater than or equal
LGT (STRING A, STRING B)	Lexically greater than
LLE (STRING A, STRING B)	Lexically less than or equal
LLT (STRING A, STRING B)	Lexically less than
MAX (A1, A2 [, A3,...])	Maximum value
MIN (A1, A2 [, A3,...])	Minimum value
REPEAT (STRING, NCOPIES)	Repeated concatenation
SCAN (STRING, SET [, BACK, KIND])	Scan a string for a character in a set
TRIM (STRING)	Remove trailing blank characters
VERIFY (STRING, SET [, BACK, KIND])	Verify the set of characters in a string

~~~~~~

13.5.4 Kind functions

KIND (X)	Kind type parameter value
SELECTED CHAR KIND (NAME)	Character kind type parameter value, given character set name
SELECTED INT KIND (R)	Integer kind type parameter value, given range
SELECTED REAL KIND ([P, R])	Real kind type parameter value, given precision and range

~~~~~~

13.5.5 Miscellaneous type conversion functions

LOGICAL (L [, KIND])	Convert between objects of type logical with different kind type parameters
TRANSFER (SOURCE, MOLD [, SIZE])	Treat first argument as if of type of second argument

~~~~~~

13.5.6 Numeric inquiry functions

DIGITS (X)	Number of significant digits of the model
EPSILON (X)	Number that is almost negligible compared to one
HUGE (X)	Largest number of the model
MAXEXPONENT (X)	Maximum exponent of the model
MINEXPONENT (X)	Minimum exponent of the model
PRECISION (X)	Decimal precision
RADIX (X)	Base of the model
RANGE (X)	Decimal exponent range
TINY (X)	Smallest positive number of the model
LBOUND (ARRAY [, DIM, KIND])	Lower dimension bounds of an array
SHAPE (SOURCE [, KIND])	Shape of an array or scalar
SIZE (ARRAY [, DIM, KIND])	Total number of elements in an array
UBOUND (ARRAY [, DIM, KIND])	Upper dimension bounds of an array

~~~~~~

13.5.8 Other inquiry functions

ALLOCATED (ARRAY) or ALLOCATED (SCALAR)	Allocation status
ASSOCIATED (POINTER [, TARGET])	Association status inquiry or comparison
BIT_SIZE (I)	Number of bits of the model
EXTENDS TYPE OF (A, MOLD)	Same dynamic type or an extension
LEN (STRING [, KIND])	Length of a character entity
NEW_LINE (A)	Newline character
PRESENT (A)	Argument presence
SAME TYPE AS (A, B)	Same dynamic type

~~~~~~

13.5.9 Bit manipulation procedures

BTEST (I, POS)	Bit testing
IAND (I, J)	Bitwise AND
IBCLR (I, POS)	Clear bit
IBITS (I, POS, LEN)	Bit extraction
IBSET (I, POS)	Set bit
IEOR (I, J)	Exclusive OR
IOR (I, J)	Inclusive OR
ISHFT (I, SHIFT)	Logical shift
ISHFTC (I, SHIFT [, SIZE])	Circular shift
MVBITS (FROM,FROMPOS,LEN,TO,TOPOS)	Copies bits from one integer to another
NOT (I)	Bitwise complement

~~~~~~

13.5.10 Floating-point manipulation functions

EXPONENT (X)	Exponent part of a model number
FRACTION (X)	Fractional part of a number
NEAREST (X, S)	Nearest different processor number in given direction
RRSPACING (X)	Reciprocal of the relative spacing of model numbers near given number
SCALE (X, I)	Multiply a real by its base to an integer power
SET EXPONENT (X, I)	Set exponent part of a number
SPACING (X)	Absolute spacing of model numbers near given

~~~~~~

13.5.11 Vector and matrix multiply functions
~~~~~~

DOT PRODUCT (VECTOR A,VECTOR B)
Dot product of two rank-one arrays
MATMUL (MATRIX A, MATRIX B)   Matrix multiplication
~~~~~~

13.5.12 Array reduction functions
ALL (MASK [, DIM])   True if all values are true
ANY (MASK [, DIM])   True if any value is true
COUNT (MASK [, DIM, KIND]) Number of true elements in an array
MAXVAL (ARRAY, DIM [, MASK]) or   Maximum value in an array
MAXVAL (ARRAY [, MASK])
MINVAL (ARRAY, DIM [, MASK]) or   Minimum value in an array
MINVAL (ARRAY [, MASK])
PRODUCT (ARRAY, DIM [, MASK]) or   Product of array elements
PRODUCT (ARRAY [, MASK])
SUM (ARRAY, DIM [, MASK]) or   Sum of array elements
SUM (ARRAY [, MASK])
~~~~~~

13.5.13 Array construction functions
CSHIFT (ARRAY, SHIFT [, DIM])   Circular shift
EOSHIFT (ARRAY, SHIFT [,BOUNDARY,DIM]) End-off shift
MERGE (TSOURCE, FSOURCE, MASK)   Merge under mask
[GFORTRAN QUICK-REFERENCE GUIDE PAGE 5]
LANGUAGE-REFERENCE:
PACK (ARRAY, MASK [, VECTOR])
Pack an array into an array of rank one under a mask
RESHAPE (SOURCE, SHAPE[, PAD,ORDER])   Reshape an array
SPREAD (SOURCE, DIM, NCOPIES)
Replicates array by adding a dimension
TRANSPOSE (MATRIX)   Transpose of an array of rank two
UNPACK (VECTOR, MASK, FIELD)
Unpack an array of rank one into an array under a mask
~~~~~~

13.5.14 Array location functions
MAXLOC (ARRAY, DIM [, MASK, KIND]) or
MAXLOC (ARRAY [, MASK,KIND])
Location of a maximum value in an array
MINLOC (ARRAY, DIM [, MASK, KIND]) or
MINLOC (ARRAY [, MASK, KIND])
Location of a minimum value in an array
~~~~~~

13.5.15 Null function
NULL ([MOLD]) Returns disassociated or unallocated result
13.5.16 Allocation transfer procedure
MOVE ALLOC (FROM, TO)
Moves an allocation from one allocatable object
~~~~~~

13.5.17 Random number subroutines
RANDOM NUMBER (HARVEST) Returns pseudorandom number
~~~~~~

13.5.18 System environment procedures
COMMAND ARGUMENT COUNT () Number of command arguments
CPU_TIME (TIME)   Obtain processor time
DATE_AND_TIME ([DATE, TIME, ZONE,VALUES])
Obtain date and time
GET_COMMAND ([COMMAND,LENGTH, STATUS])
Returns entire command
GET_COMMAND_ARGUMENT (NUMBER [, VALUE, LENGTH, STATUS])   Returns a command argument
GET_ENVIRONMENT_VARIABLE (NAME [,VALUE,LENGTH, STATUS,TRIM NAME])   Obtain the value of an environment variable
IS_IOSTAT_END (I)   Test for end-of-file value
IS_IOSTAT_EOR (I)   Test for end-of-record value
SYSTEM CLOCK ([COUNT,COUNT RATE, COUNT MAX])
Obtain data from the system clock
~~~~~~

New in Fortran 95:
Miscellaneous
free source form
enhancements to fixed source form:
";" statement separator
"!" trailing comment
names may be up to 31 characters in length
both upper and lower case characters are accepted
INCLUDE line
relational operators in mathematical notation
enhanced END statement
IMPLICIT NONE
binary, octal, and hexadecimal constants
quotation marks around CHARACTER constants
Data
enhanced type declaration statements
new attributes:
extended DIMENSION attribute
ALLOCATABLE   POINTER   TARGET   INTENT
PUBLIC   PRIVATE
kind and length type parameters   derived types
pointers
Operations
extended intrinsic operators   extended assignment
user-defined operators
Arrays
automatic arrays   allocatable arrays
New in Fortran 95 (Continued):
assumed-shape arrays   array sections
array expressions
masked array assignment (WHERE statement and construct)
FORALL statement*
Execution Control
LANGUAGE-REFERENCE continued (work-in-progress):
CASE construct   enhance DO construct
CYCLE statement   EXIT statement
Input/Output
binary, octal, and hexadecimal edit descriptors
engineering and scientific edit descriptors
namelist formatting
partial record capabilities (non-advancing I/O)
extra OPEN and INQUIRE specifiers
Procedures
keyword arguments   optional arguments
INTENT attribute   derived type actual arguments and functions
array-valued functions   recursive procedures
user-defined generic procedures
user-defined elemental procedures*   pure procedures*
specification of procedure interfaces   internal procedures
Modules
New Intrinsic Procedures
NULL*   PRESENT   (See manual for List)
~~~~~~