Fortran 90 Reference Card

(c) 2008 Michael Goerz <goerz@physik.fu-berlin.de>
http://www.michaelgoerz.net

For a complete reference, use Adams et al., *Fortran 90 Handbook*, Intertext Publications, 1992.

1 Data Types

1.1 Simple Data Types (entity-oriented declarations)

`integer(specs) [,attrs] :: i=1`	integer (with initialization)
`real(specs) [,attrs] :: r`	real number
`complex(specs) [,attrs] :: z`	complex number
`logical(specs) [,attrs] :: b`	boolean variable
`character(specs) [,attrs] :: s`	string
`data i,j,k/3*0/`	initialize i, j, k to 0
`kind(x)`	kind-parameter of variable x
`real, parameter :: c = 2.998`	constant declaration
`bitsize(i)`	number of bits for int

attributes: `parameter, pointer, target, allocatable, dimension, public, private, intent, optional, save, external, intrinsic`

specs: `kind=...,` for character: `len=...`

1.1 Derived Data Types

`type person`	Define person as derived data type
`  character(len=10) :: name`	
`  integer :: age`	
`end type person`	
`type(person) :: me`	instantiate person
`me = person(“michael”, 24)`	constructor
`name = me%name`	access member

1.2 Arrays and Matrices

`real, dimension(5) :: v`	explicit array with index 1..5
`real, dimension(-1:1,3) :: a`	2D array, index -1..1, 1..3
`integer :: a(-10:5), b(10,20)`	alternative array declaration
`real, allocatable :: a(:)`	alloc. array ("deferred shape")
`a=real(5,5); data a/25*0.0/`	initialize 2D array
`v = 1/v + a(1:5,5)`	array expression
`allocate(a(5),b(2:4),stat=e)`	array allocation

1.3 Pointers (avoid)

`real, pointer :: p`	declare pointer
`real, pointer :: a(:)`	alloc. array ("deferred shape")
`real, target :: r`	define target
`p => r`	set pointer p to `r`
`associated(p, [target])`	pointer associated with target?
`nullify(p)`	associate pointer with NUL

1.4 Operators

`.lt. .le. .eq. .ne. .gt. .ge.`	relational operators
`.not. .and. .or. .eqv. .neqv.`	logical operators
`x**(-y)`	exponentiation
`'AB'//'CD'`	string concatenation

2 Control Constructs

`goto 10`	go to label 10
`if (expr) action`	if statement
`[name:] if (expr) then`	if-construct
`  block`	
`else if (expr) then [name]`	
`  block`	
`else [name]`	
`  block`	
`end if [name]`	
`select case (number)`	select-statement
`  case (:0); block`	everything up to 0 (incl.)
`  case (1:); block`	everything up from 1 (incl.)
`end select`	
`outer: do`	controlled do-loop
`  inner: do i=from,to,step`	counter do-loop
`    if (...) cycle inner`	next iteration
`    if (...) exit outer`	exit from named loop
`  end do inner`	
`end do outer`	
`do while (expr)`	do-while loop
`  block`	
`end do`	

3 Program Structure

`program foo`	main program
`  use foo, lname => usename`	used module, with rename
`  use foo2, only: [only-list]`	selective use
`  implicit none`	require variable declaration
`  interface; ... end interface`	explicit interfaces
`  specification statements`	variable/type declarations, etc.
`  exec statements`	statements
`  stop 'message'`	terminate program
`contains`	
`  internal-subprograms`	subroutines, functions
`end program foo`	
`module foo`	module
`  use foo`	used module
`  public :: f1, f2, ...`	list public subroutines
`  private`	make private what's not public
`  interface; ... end interface`	explicit interfaces
`  specification statements`	variable/type declarations, etc.
`contains`	
`  internal-subprograms`	“ module subprgs.”
`end module foo`	
`subroutine foo(a,b,c,d,e,x,y)`	subroutine definition
`  integer, intent(in) :: a`	read-only dummy variable
`  integer, intent(inout) :: b`	read-write dummy variable
`  integer, intent(out) :: c`	write-only dummy variable
`  real, optional :: d`	optional named argument
`  character (len=*) :: e`	assumed length string
`  real, dimension (2:, :) :: x`	assumed-shape dummy array
`  real, dimension (10, *) :: y`	assumed-size dummy array
`  if (present(d)) ...`	presence check
`  return`	forced exit
`end subroutine foo`	
`[real] function f(a,g)`	function definition
`  integer, intent(in) :: a`	input parameter
`  [real :: f]`	return type, if not in definition
`  interface`	explicit interface block
`    real function g(x)`	define dummy var as function
`      real, intent(in) :: x`	
`    end function g`	
`  end interface`	
`end function f`	
`recursive function f(x) ...`	allow recursion
`incr(x) = x + 1`	statement function
`interface`	explicit interface of externals
`  interface body`	ext. subroutine/function specs
`end interface`	
`interface generic-name`	generic interface (overloading)
`  interface body`	external subroutines/functions
`  module procedure list`	internal subroutines/functions
`end interface`	
`interface operator op`	operator interface
`  interface body`	external functions
`  module procedure list`	internal functions
`end interface`	
`interface assignment (=)`	conversion interface
`  interface body`	external subroutines
`  module procedure list`	internal subroutines
`end interface`	

4 Intrinsic Procedures

4.1 Transfer and Conversion Functions

`abs(a)`	absolute value
`aimag(z)`	imaginary part of complex z
`aint(x, kind), anint(x, kind)`	to whole number real
`dble(a)`	to double precision
`cmplx(x,y, kind)`	create x + iy (y optional)
`int(a, kind), nint(a, kind)`	to int (truncated/rounded)
`real(x, kind)`	to real
`conj(z)`	complex conjugate
`char(i, kind), achar(i)`	char of ASCII code (pure 7bit)
`ichar(c), iachar(c)`	ASCII code of character
`logical(l, kind)`	change kind of logical l
`ibits(i, pos, len)`	extract sequence of bits
`transfer(source, mold, size)`	reinterpret data

4.2 Arrays and Matrices

`allocated(a)`	check if array is allocated
`lbound(a, dim), ubound(a,dim)`	lowest/highest index in array
`shape(a)`	shape (dimensions) of array
`size(array, dim)`	extent of array along dim
`all(mask, dim), any(mask, dim)`	check boolean array
`count(mask, dim)`	number of true elements
`maxval(a,d,m), minval(a,d,m)`	find max/min in masked array
`product(a, dim, mask)`	product along masked dimen.
`sum(array, dim, mask)`	sum along masked dimension
`merge(tsource, fsource, mask)`	combine arrays as mask says
`pack(array, mask, vector)`	packs masked array into vect.
`unpack(vector, mask , field)`	unpack vect. into masked field
`spread(source, dim, n)`	extend source array into dim.
`reshape(src,shape,pad,order)`	make array of shape from src
`cshift(a,s,d),eoshift(a,s,b,d)`	(circular) shift

transpose(matrix)	transpose a matrix
maxloc(a,mask), minloc(a,mask)	find pos. of max/min in array

4.3 Computation Functions

ceiling(a), floor(a)	to next higher/lower int
conj(z)	complex conjugate
dim(x,y)	max(x-y, 0)
max(a1, a2, a3..), min(a1, ..)	maximum/minimum
dprod(a,b)	dp product of sp a, b
mod(a,p), modulo(a,p)	modulo (having sign of a / p)
sign(a,b)	make sign of a = sign of b
matmul(m1, m2)	matrix multiplication
dot_product(a,b)	dot product of vectors

more: sin, cos, tan, acos, asin, atan, atan2, sinh, cosh, tanh, exp, log, log10, sqrt

4.4 String Functions

lge(s1,s2), lgt, lle, llt	string comparison
adjustl(s), adjustr(s)	left- or right-justify string
index(s, sub, from_back)	find substr. in string (or 0)
trim(s)	s without trailing blanks
len_trim(s)	length of s, w/ trailing blanks
scan(s, setd, from_back)	search for any char in set
verify(s, set, from_back)	check for presence of set-chars
len(string)	length of string
repeat(string, n)	concat n copies of string

4.5 Bit Functions (on integers)

btest(i,pos)	test bit of integer value
iand(i,j), ieor(i,j), ior(i,j)	and, xor, or of bit in 2 integers
ibclr(i,pos), ibset(i, pos)	set bit of integer to 0 / 1
ishft(i, sh), ishftc(i, sh, s)	shift bits in i
not(i)	bit-reverse integer

4.6 Misc Intrinsic Subroutines

date_and_time(d, t, z, v)	put current time in d,t,z,v
mvbits(f, fpos, len, t, tpos)	copy bits between int vars
random_number(harvest)	fill harvest randomly
random_seed(size, put, get)	restart/query random generator
system_clock(c, cr, cm)	get processor clock info

numeric inquiry functions: digits, epsilon, huge, minexponent, maxeponent, precision, radix, range, tiny

numeric manipulation functions: exponent, fraction, nearest, rrspacing, scale, set_exponent, spacing

5 Input/Output

5.1 Format Statements

fmt = "(F10.3, A, ES14.7)"	format string
I*w* I*w*.*m*	integer form
Bw.m Ow.m Zw.m	binary, octal, hex integer form
F*w*.*d*	decimal form real format
Ew.d	exponential form (0.12..E-11)
E*w*.*d*E*e*	specified exponent length
ESw.d ES*w*.*d*E*e*	scientific form (1.2...E-10)
ENw.d EN*w*.*d*E*e*	engineer. form (123.4...E-12)
Gw.d	generalized form
Gw.dEe	generalized exponent form
L*w*	logical format (T, F)
fmt = "(F10.3, A, ES14.7)"	format string
A A*w*	characters format
*n*X	horizontal positioning (skip)
T*c* TL*c* TR*c*	move (absolute, left, right)
r/	vert. positioning (skip lines)
r(...)	grouping / repetition
:	format scanning control
S SP SS	sign control
BN BZ	blank control (blanks as zeros)

w full length, *m* minimum digits, *d* decimal places, *e* exponent length, *n* positions to skip, *c* positions to move, *r* repetitions

5.2 Reading from and Writing to Files

call getarg(2, *var*)	put 2nd CLI-argument in var
print '(i10)', 2	print to stdout with format
print *, “Hello World”	list-directed I/O
write(*unit*, *fmt*, *spec*) list	write list to unit
read(*unit*, fmt) list	read list from unit
open(*unit*, *specifiers*)	open file (see below)
close(*unit*, *specifiers*)	
inquire	
backspace	
endfile	
rewind	

5.3 Specifiers (open)

iostat=*integer-variable*	save iocode to variable
err=*errorlabel*	label to jump to on error
file='*filename*'	name of file to open
status='old' 'new' 'replace' 'scratch' 'unknown'	status of input file
access='sequential' 'direct'	access method
form='formatted' 'unformatted'	formatted/unformatted I/O
recl=*integer*	length of record
blank='null' 'zero'	ignore blanks/treat them as 0
position='asis''rewind' 'append'	position, if sequential I/O
action='read' 'write' 'readwrite'	read/write mode
delim='quote' 'apostrophe' 'none'	delimiter for char constants
pad='yes' 'no'	pad with blanks

close-specifiers: iostat, err, status='keep' 'delete'