

Fortran 90 Reference Card

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

For a complete reference, I highly recommend
Adams, Brainerd, Martin, Smith, Wagener, *Fortran 90 Handbook*, Intertext Publications, 1992.

This work is licensed under the Creative Commons Attribution-Noncommercial-Share Alike 3.0 License.
To view a copy of this license, visit <http://creativecommons.org/licenses/by-nc-sa/>

1 Data Types

1.1 Simple Data Types (entity-oriented declarations)

integer(<i>specs</i>) [<i>,attrs</i>] :: i=1	integer (with initialization)
real(<i>specs</i>) [<i>,attrs</i>] :: r	real number
complex(<i>specs</i>) [<i>,attrs</i>] :: z	complex number
logical(<i>specs</i>) [<i>,attrs</i>] :: b	boolean variable
character(<i>specs</i>) [<i>,attrs</i>] :: s	string
real, parameter :: c = 2.998	constant declaration
data i,j,k/3*0/	initialize i,j,k to 0
s2=s(2:5); s2=s(:5); s2=s(5:)	substring extraction

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
character(len=10) :: name	type
integer :: age	
end type person	
type(person) :: me	instantiate person
me = person("michael", 24)	constructor
name = me%name	access structure component

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
a=(/1.2,b(2:6,:),3.5/)	array constructor
a=(/(I**2), I = 1, N)/)	implied-do array constructor
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

if (<i>expr</i>) <i>action</i>	if statement
[<i>name</i> :] if (<i>expr</i>) then	if-construct
block	
else if (<i>expr</i>) then [<i>name</i>]	
block	
else [<i>name</i>]	
block	
end if [<i>name</i>]	
select case (number)	select-statement
case (:0)	everything up to 0 (incl.)
block	
case (1:2)	number is 1 or 2
block	
case (3)	number is 3
block	
case (4:)	everything up from 4 (incl.)
block	
case default	fall-through case
block	
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 (<i>expr</i>)	do-while loop
block	
end do	

3 Program Structure

program foo	main program
use foo, lname => username	used module, with rename
use foo2, only: [only-list]	selective use
implicit none	require variable declaration
interface; ...	explicit interfaces
end interface	
specification statements	variable/type declarations, etc.
exec statements	statements
stop 'message'	terminate program
contains	
internal-subprograms	
end program foo	subroutines, functions
module foo	module
use foo	used module
public :: f1, f2, ...	list public subroutines
private	make private what's not public
interface; ...	explicit interfaces
end interface	
specification statements	variable/type declarations, etc.
contains	
internal-subprograms	
end module foo	" module subprgs."

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	
call foo(1,2,3,e="s",x=a,y=b)	subroutine call
[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)
lbound(a, dim), ubound(a,dim)
shape(a)
size(array, dim)
all(mask, dim), any(mask, dim)
count(mask, dim)
maxval(a,d,m), minval(a,d,m)
product(a, dim, mask)
sum(array, dim, mask)
merge(tsource, fsource, mask)
pack(array, mask, vector)
unpack(vector, mask, field)
spread(source, dim, n)
reshape(src, shape, pad, order)
cshift(a,s,d), eoshift(a,s,b,d)
transpose(matrix)
maxloc(a,mask), minloc(a,mask)

check if array is allocated
lowest/highest index in array
shape (dimensions) of array
extent of array along dim
check boolean array
number of true elements
find max/min in masked array
product along masked dimen.
sum along masked dimension
combine arrays as mask says
packs masked array into vect.
unpack vect. into masked field
extend source array into dim.
make array of shape from src
(circular) shift
transpose a matrix
find pos. of max/min in array

4.3 Computation Functions

ceiling(a), floor(a)
conj(z)
dim(x,y)
max(a1, a2, a3...), min(a1, ..)
dprod(a,b)
mod(a,p)
modulo(a,p)
sign(a,b)
matmul(m1, m2)
dot_product(a,b)

to next higher/lower int
complex conjugate
max(x-y, 0)
maximum/minimum
dp product of sp a, b
a mod p
modulo with sign of a/p
make sign of a = sign of b
matrix multiplication
dot product of vectors

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

4.4 Numeric Inquiry and Manipulation Functions

kind(x)
digits(x)
bit_size(i)
epsilon(x)
huge(x)
minexponent(x)
maxexponent(x)
precision(x)
radix(x)
range(x)
tiny(x)
exponent(x)
fraction(x)
nearest(x)
rrspacing(x)
scale(x,i)
set_exponent(x,i)
spacing(x)

kind-parameter of variable x
significant digits in model
number of bits for int in model
small pos. number in model
largest number in model
smallest exponent in model
largest exponent in model
decimal precision for reals in
base of the model
dec. exponent range in model
smallest positive number
exponent part of x in model
fractional part of x in model
nearest machine number
reciprocal of relative spacing
x b**i
x b**(i-e)
absolute spacing of model

4.5 String Functions

lge(s1,s2), lgt, lle, llt
adjustl(s), adjustr(s)
index(s, sub, from_back)
trim(s)

string comparison
left- or right-justify string
find substr. in string (or 0)
s without trailing blanks

len_trim(s)
scan(s, setd, from_back)
verify(s, set, from_back)
len(string)
repeat(string, n)

4.6 Bit Functions (on integers)

btest(i,pos)
iand(i,j), ieor(i,j), ior(i,j)
ibclr(i,pos), ibset(i, pos)
ishft(i, sh), ishftc(i, sh, s)
not(i)

4.7 Misc Intrinsic Subroutines

date_and_time(d, t, z, v)
mvbits(f, fpos, len, t, tpos)
random_number(harvest)
random_seed(size, put, get)
system_clock(c, cr, cm)

length of s, w/ trailing blanks
search for any char in set
check for presence of set-chars
length of string
concat n copies of string

test bit of integer value
and, xor, or of bit in 2 integers
set bit of integer to 0 / 1
shift bits in i
bit-reverse integer

put current time in d,t,z,v
copy bits between int vars
fill harvest randomly
restart/query random generator
get processor clock info

5 Input/Output

5.1 Format Statements

fmt = "(F10.3, A, ES14.7)"
Iw Iw.m
Bw.m Ow.m Zw.m
Fw.d
Ew.d
Ew.dEe
ESw.d ESw.dEe
ENw.d ENw.dEe
Gw.d
Gw.dEe
Lw
A Aw
nX
Tc Tlc TRc
r/
r(...)
:
S SP SS
BN BZ

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)
print '(i10)', 2
print *, "Hello World"
write(unit, fmt, spec) list
read(unit, fmt, spec) list
open(unit, specifiers)
close(unit, specifiers)
inquire(unit, spec)
inquire(file=filename, spec)
inquire(iolength=iol) outlist
backspace(unit, spec)
endfile(unit, spec)
rewind(unit, spec)

format string
integer form
binary, octal, hex integer form
decimal form real format
exponential form (0.12..E-11)
specified exponent length
scientific form (1.2...E-10)
engineer. form (123.4...E-12)
generalized form
generalized exponent form
logical format (T, F)
characters format
horizontal positioning (skip)
move (absolute, left, right)
vert. positioning (skip lines)
grouping / repetition
format scanning control
sign control
blank control (blanks as zeros)

put 2nd CLI-argument in var
print to stdout with format
list-directed I/O
write list to unit
read list from unit
open file
close file
inquiry by unit
inquiry by filename
inquiry by output item list
go back one record
write eof record
jump to beginning of file

5.3 I/O Specifiers (open)

iostat=*integer-variable*
err=*label*
file=*filename*
status='old' 'new' 'replace'
'scratch' 'unknown'
access='sequential' 'direct'
form='formatted' 'unformatted'
recl=*integer*
blank='null' 'zero'
position='asis' 'rewind'
'append'
action='read' 'write'
'readwrite'
delim='quote' 'apostrophe'
'none'
pad='yes' 'no'
close-specifiers: iostat, err, status='keep' 'delete'
inquire-specifiers: access, action, blank, delim, direct,
exist, form, formatted, iostat, name, named, nextrec,
number, opened, pad, position, read, readwrite, recl,
sequential, unformatted, write, iolength
backspace-, endfile-, rewind-specifiers: iostat, err

save iocode (error) to variable
label to jump to on error
name of file to open
status of input file

access method
formatted/unformatted I/O
length of record
ignore blanks/treat them as 0
position, if sequential I/O

read/write mode
delimiter for char constants

pad with blanks

5.4 Data-Transfer Specifiers (read, write)

iostat=*integer-variable*
advance='yes' 'no'
err=*label*
end=*label*
eor=*label*
rec=*integer*
size=*integer-variable*

save iocode (error) to variable
(non-)advancing data transfer
label to jump to on error
label to jump to on end of file
label for end of record
record number to read or write
number of characters read