Appendix B

Fortran 90 Overview

This overview of Fortran 90 (F90) features is presented as a series of tables that illustrate the syntax and abilities of F90. Frequently comparisons are made to similar features in the C++ and F77 languages and to the Matlab environment.

These tables show that F90 has significant improvements over F77 and matches or exceeds newer software capabilities found in C++ and Matlab for dynamic memory management, user defined data structures, matrix operations, operator definition and overloading, intrinsics for vector and parallel processors and the basic requirements for object-oriented programming.

They are intended to serve as a condensed quick reference guide for programming in F90 and for understanding programs developed by others.

B.1 List of Language Tables

Language	Syntax	Location
MATLAB	`% comment (to end of line)`	anywhere
C	`/*comment*/`	anywhere
F90	`! comment (to end of line)`	anywhere
F77	`* comment (to end of line)`	column 1

Table B.1: Comment syntax.

Storage	MATLAB[a]	C++	F90	F77
byte		`char`	`character::`	`character`
integer		`int`	`integer::`	`integer`
single precision		`float`	`real::`	`real`
double precision		`double`	`real*8::`	`double precision`
complex		[b]	`complex::`	`complex`
Boolean		`bool`	`logical::`	`logical`
argument			`parameter::`	`parameter`
pointer		`*`	`pointer::`	
structure		`struct`	`type::`	

[a] MATLAB4 requires no variable type declaration; the only two distinct types in MATLAB are strings and reals (which include complex). Booleans are just 0s and 1s treated as reals. MATLAB5 allows the user to select more types.

[b] There is no specific data type for a complex variable in C++; they must be created by the programmer.

Table B.2: Intrinsic data types of variables.

Description	MATLAB[a]	C++	Fortran[b]
addition	`+`	`+`	`+`
subtraction[c]	`-`	`-`	`-`
multiplication	`*` and `.*`	`*`	`*`
division	`/` and `./`	`/`	`/`
exponentiation	`^` and `.^`	pow[d]	`**`
remainder		`%`	
increment		`++`	
decrement		`--`	
parentheses (expression grouping)	`()`	`()`	`()`

[a] When doing arithmetic operations on matrices in MATLAB, a period (‘`.`’) must be put before the operator if scalar arithmetic is desired. Otherwise, MATLAB assumes matrix operations; figure out the difference between ‘`*`’ and ‘`.*`’. Note that since matrix and scalar addition coincide, no ‘`.+`’ operator exists (same holds for subtraction).

[b] Fortran 90 allows the user to change operators and to define new operator symbols.

[c] In all languages the minus sign is used for negation (i.e., changing sign).

[d] In C++ the exponentiation x^y is calculated by function $pow\,(x, y)$.

Table B.3: Arithmetic operators.

Description	MATLAB	C++	F90	F77
Equal to	==	==	==	.EQ.
Not equal to	~=	!=	/=	.NE.
Less than	<	<	<	.LT.
Less or equal	<=	<=	<=	.LE.
Greater than	>	>	>	.GT.
Greater or equal	>=	>=	>=	.GE.
Logical NOT	~	!	.NOT.	.NOT.
Logical AND	&	&&	.AND.	.AND.
Logical inclusive OR	!	||	.OR.	.OR.
Logical exclusive OR	xor		.XOR.	.XOR.
Logical equivalent	==	==	.EQV.	.EQV.
Logical not equivalent	~=	!=	.NEQV.	.NEQV.

Table B.4: Relational operators (arithmetic and logical).

MATLAB Operators	C++ Operators	F90 Operators[a]	F77 Operators
()	() [] -> .	()	()
+ -	! ++ -- + - * & (type) sizeof	**	**
* /	* / %	* /	* /
+ -[b]	+ -[b]	+ -[b]	+ -[b]
< <= > >=	<< >>	//	//
== ~=	< <= > =>	== /= < <= > >=	.EQ. .NE. .LT. .LE. .GT. .GE.
~	== !=	.NOT.	.NOT.
&	&&	.AND.	.AND.
|	||	.OR.	.OR.
=	|	.EQV. .NEQV.	.EQV. .NEQV.
	?:		
	= += -= *= /= %= &= ^= |= <<= >>=		
	,		

[a] User-defined unary (binary) operators have the highest (lowest) precedence in F90.
[b] These are binary operators representing addition and subtraction. Unary operators + and - have higher precedence.

Table B.5: Precedence pecking order.

B = Beginning, E = Ending, I = Increment

Syntax	F90	MATLAB
Default	B:E:I	B:I:E
$\geq$ B	B:	B:
$\leq$ E	:E	:E
Full range	:	:

Use	F90	MATLAB
Array subscript ranges	yes	yes
Character positions in a string	yes	yes
Loop control	no	yes
Array element generation	no	yes

Table B.6: Colon Operator Syntax and its Applications.

Description	MATLAB	C++	F90	F77
exponential	`exp(x)`	`exp(x)`	`exp(x)`	`exp(x)`
natural log	`log(x)`	`log(x)`	`log(x)`	`log(x)`
base 10 log	`log10(x)`	`log10(x)`	`log10(x)`	`log10(x)`
square root	`sqrt(x)`	`sqrt(x)`	`sqrt(x)`	`sqrt(x)`
raise to power (x^r)	`x.^r`	`pow(x,r)`	`x**r`	`x**r`
absolute value	`abs(x)`	`fabs(x)`	`abs(x)`	`abs(x)`
smallest integer>x	`ceil(x)`	`ceil(x)`	`ceiling(x)`	
largest integer<x	`floor(x)`	`floor(x)`	`floor(x)`	
division remainder	`rem(x,y)`	`fmod(x,y)`	`mod(x,y)`[a]	`mod(x,y)`
modulo			`modulo(x,y)`[a]	
complex conjugate	`conj(z)`		`conjg(z)`	`conjg(z)`
imaginary part	`imag(z)`		`imag(z)`	`aimag(z)`
drop fraction	`fix(x)`		`aint(x)`	`aint(x)`
round number	`round(x)`		`nint(x)`	`nint(x)`
cosine	`cos(x)`	`cos(x)`	`cos(x)`	`cos(x)`
sine	`sin(x)`	`sin(x)`	`sin(x)`	`sin(x)`
tangent	`tan(x)`	`tan(x)`	`tan(x)`	`tan(x)`
arc cosine	`acos(x)`	`acos(x)`	`acos(x)`	`acos(x)`
arc sine	`asin(x)`	`asin(x)`	`asin(x)`	`asin(x)`
arc tangent	`atan(x)`	`atan(x)`	`atan(x)`	`atan(x)`
arc tangent[b]	`atan2(x,y)`	`atan2(x,y)`	`atan2(x,y)`	`atan2(x,y)`
hyperbolic cosine	`cosh(x)`	`cosh(x)`	`cosh(x)`	`cosh(x)`
hyperbolic sine	`sinh(x)`	`sinh(x)`	`sinh(x)`	`sinh(x)`
hyperbolic tangent	`tanh(x)`	`tanh(x)`	`tanh(x)`	`tanh(x)`
hyperbolic arc cosine	`acosh(x)`			
hyperbolic arc sine	`asinh(x)`			
hyperbolic arctan	`atanh(x)`			

[a] Differ for $x < 0$.

[b] `atan2(x,y)` is used to calculate the arc tangent of x/y in the range $[-\pi, +\pi]$. The one-argument function `atan(x)` computes the arc tangent of x in the range $[-\pi/2, +\pi/2]$.

Table B.7: Mathematical functions.

Description	C++	F90	F77	MATLAB
Conditionally execute statements	`if` `{ }`	`if` `end if`	`if` `end if`	`if` `end`
Loop a specific number of times	`for k=1:n` `{ }`	`do k=1,n` `end do`	`do # k=1,n` `# continue`	`for k=1:n` `end`
Loop an indefinite number of times	`while` `{ }`	`do while` `end do`	— —	`while` `end`
Terminate and exit loop	`break`	`exit`	`go to`	`break`
Skip a cycle of loop	`continue`	`cycle`	`go to`	—
Display message and abort	`error()`	`stop`	`stop`	`error`
Return to invoking function	`return`	`return`	`return`	`return`
Conditional array action	—	`where`	—	`if`
Conditional alternate statements	`else` `else if`	`else` `elseif`	`else` `elseif`	`else` `elseif`
Conditional array alternatives	— —	`elsewhere` —	— —	`else` `elseif`
Conditional case selections	`switch { }`	`select case` `end select`	`if` `end if`	`if` `end`

Table B.8: Flow Control Statements.

Loop	MATLAB	C++	Fortran
Indexed loop	`for index=matrix` `  statements` `end`	`for (init;test;inc)` `  {` `  statements` `  }`	`do index=b,e,i` `  statements` `  end do`
Pre-test loop	`while test` `  statements` `end`	`while (test) {` `  statements` `  }`	`do while (test)` `  statements` `  end do`
Post-test loop		`do {` `  statements` `  } while (test)`	`do` `  statements` `  if (test) exit` `  end do`

Table B.9: Basic loop constructs.

MATLAB	Fortran	C++
if l_expression *true group* end	IF (l_expression) THEN *true group* END IF	if (l_expression) { *true group;* }
	IF (l_expression) *true statement*	if (l_expression) *true statement;*

Table B.10: IF Constructs. The quantity l_expression means a logical expression having a value that is either TRUE of FALSE. The term *true statement* or *true group* means that the statement or group of statements, respectively, are executed if the conditional in the if statement evaluates to TRUE.

MATLAB	Fortran	C++
if l_expression1 *true group A* if l_expression2 *true group B* end *true group C* end *statement group D*	IF (l_expression1) THEN *true group A* IF (l_expression2) THEN *true group B* END IF *true group C* END IF *statement group D*	if (l_expression1) { true group A if (l_expression2) { *true group B* } *true group C* } *statement group D*

Table B.11: Nested IF Constructs.

MATLAB	Fortran	C++
if l_expression *true group A* else *false group B* end	IF (l_expression) THEN *true group A* ELSE *false group B* END IF	if (l_expression) { *true group A* } else { *false group B* }

Table B.12: Logical IF-ELSE Constructs.

MATLAB	Fortran	C++
if l_expression1 *true group A* elseif l_expression2 *true group B* elseif l_expression3 *true group C* else *default group D* end	IF (l_expression1) THEN *true group A* ELSE IF (l_expression2) THEN *true group B* ELSE IF (l_expression3) THEN *true group C* ELSE *default group D* END IF	if (l_expression1) { *true group A* } else if (l_expression2) { *true group B* } else if (l_expression3) { *true group C* } else { *default group D* }

Table B.13: Logical IF-ELSE-IF Constructs.

F90	C++
SELECT CASE (expression) CASE (value 1) *group 1* CASE (value 2) *group 2* : CASE (value n) *group n* CASE DEFAULT *default group* END SELECT	switch (expression) { case value 1 : *group 1* break; case value 2 : *group 2* break; : case value n : *group n* break; default: *default group* break; }

Table B.14: Case Selection Constructs.

F90 Named `IF`	F90Named `SELECT`
name: IF (logical_1) THEN *true group A* ELSE IF (logical_2) THEN *true group B* ELSE *default group C* ENDIF name	name: SELECT CASE (expression) CASE (value 1) *group 1* CASE (value 2) *group 2* CASE DEFAULT *default group* END SELECT name

Table B.15: F90 Optional Logic Block Names.

Fortran	C++
DO 1 ... DO 2 ... ... IF (disaster) THEN GO TO 3 END IF ... 2 END DO 1 END DO 3 *next statement*	for (...) { for (...) { ... if (disaster) go to error ... } } error:

Table B.16: `GO TO` Break-out of Nested Loops. This situation can be an exception to the general recommendation to avoid `GO TO` statements.

F77	F90	C++
DO 1 I = 1,N ... IF (skip condition) THEN GO TO 1 ELSE *false group* END IF 1 continue	DO I = 1,N ... IF (skip condition) THEN CYCLE ! to next I ELSE *false group* END IF END DO	for (i=1; i<n; i++) { if (skip condition) continue; // to next else if *false group* end }

Table B.17: Skip a Single Loop Cycle.

F77	F90	C++
DO 1 I = 1,N IF (exit condition) THEN GO TO 2 ELSE *false group* END IF 1 CONTINUE 2 *next statement*	DO I = 1,N IF (exit condition) THEN EXIT ! this do ELSE *false group* END IF END DO *next statement*	for (i=1; i<n; i++) { if (exit condition) break;// out of loop else if *false group* end } *next statement*

Table B.18: Abstract a Single Loop.

```
main: DO ! forever
  test: DO k=1,k_max
    third: DO m=m_max,m_min,-1
      IF (test condition) THEN
        CYCLE test ! loop on k
      END IF
    END DO third ! loop on m
    fourth: DO n=n_min,n_max,2
      IF (main condition) THEN
        EXIT main ! forever loop
    END DO fourth ! on n
  END DO test ! over k
END DO main
next statement
```

Table B.19: F90 DOs Named for Control.

MATLAB	C++
initialize test while l_expression true group change test end	*initialize test* while (l_expression) { *true group* *change test* }

F77	F90
initialize test # continue IF (l_expression) THEN *true group* *change test* go to # END IF	initialize test do while (l_expression) *true group* *change test* end do

Table B.20: Looping While a Condition is True.

Function Type	MATLAB[a]	C++	Fortran
program	*statements* [y1...yn]=f(a1,...,am) [end of file]	main(argc,char **argv) { *statements* y = f(a1,I,am); }	program main type y type a1,...,type am *statements* y = f(a1,...,am) call s(a1,...,am) end program
subroutine		void f (type a1,...,type am) { *statements* }	subroutine s(a1,...,am) type a1,...,type am *statements* end
function	function [r1...rn] =f(a1,...,am) *statements*	type f (type a1,...,type am) { *statements* }	function f(a1,...,am) type f type a1,...,type am *statements* end

[a] Every function or program in MATLAB must be in separate files.

Table B.21: Function definitions. In each case, the function being defined is named `f` and is called with `m` arguments `a1,...,am`.

One-Input, One-Result Procedures	
MATLAB	function out = name (in)
F90	function name (in) ! name = out
	function name (in) result (out)
C++	name (in, out)

Multiple-Input, Multiple-Result Procedures	
MATLAB	function [inout, out2] = name (in1, in2, inout)
F90	subroutine name (in1, in2, inout, out2)
C++	name(in1, in2, inout, out2)

Table B.22: Arguments and return values of subprograms.

Global Variable Declaration	
MATLAB	`global` list of variables
F77	`common` /set_name/ list of variables
F90	`module` set_name `save` `type` (type_tag) :: list of variables `end module` set_name
C++	`extern` list of variables

Access to Global Variables	
MATLAB	`global` list of variables
F77	`common` /set_name/ list of variables
F90	`use` set_name, `only` subset of variables `use` set_name2 list of variables
C++	`extern` list of variables

Table B.23: Defining and referring to global variables.

Action	C++	F90
Bitwise AND	&	iand
Bitwise exclusive OR	∧	ieor
Bitwise exclusive OR	|	ior
Circular bit shift		ishftc
Clear bit		ibclr
Combination of bits		mvbits
Extract bit		ibits
Logical complement	∼	not
Number of bits in integer	sizeof	bit_size
Set bit		ibset
Shift bit left	≪	ishft
Shift bit right	≫	ishft
Test on or off		btest
Transfer bits to integer		transfer

Table B.24: Bit Function Intrinsics.

```
  0 NUL   1 SOH   2 STX   3 ETX   4 EOT   5 ENQ   6 ACK   7 BEL
  8 BS    9 HT   10 NL   11 VT   12 NP   13 CR   14 SO   15 SI
 16 DLE  17 DC1  18 DC2  19 DC3  20 DC4  21 NAK  22 SYN  23 ETB
 24 CAN  25 EM   26 SUB  27 ESC  28 FS   29 GS   30 RS   31 US
 32 SP   33  !   34  "   35  #   36  $   37  %   38  &   39  ’
 40  (   41  )   42  *   43  +   44  ,   45  -   46  .   47  /
 48  0   49  1   50  2   51  3   52  4   53  5   54  6   55  7
 56  8   57  9   58  :   59  ;   60  <   61  =   62  >   63  ?
 64  @   65  A   66  B   67  C   68  D   69  E   70  F   71  G
 72  H   73  I   74  J   75  K   76  L   77  M   78  N   79  O
 80  P   81  Q   82  R   83  S   84  T   85  U   86  V   87  W
 88  X   89  Y   90  Z   91  [   92  \   93  ]   94  ^   95  _
 96  ‘   97  a   98  b   99  c  100  d  101  e  102  f  103  g
104  h  105  i  106  j  107  k  108  l  109  m  110  n  111  o
112  p  113  q  114  r  115  s  116  t  117  u  118  v  119  w
120  x  121  y  122  z  123  {  124  |  125  }  126  ~  127 DEL
```

Table B.25: The ACSII Character Set.

`ACHAR (I)`	Character number I in ASCII collating set
`ADJUSTL (STRING)`	Adjust left
`ADJUSTR (STRING)`	Adjust right
`CHAR (I)` *	Character I in processor collating set
`IACHAR (C)`	Position of C in ASCII collating set
`ICHAR (C)`	Position of C in processor collating set
`INDEX (STRING, SUBSTRING)`[a]	Starting position of a substring
`LEN (STRING)`	Length of a character entity
`LEN_TRIM (STRING)`	Length without trailing blanks
`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
`REPEAT (STRING, NCOPIES)`	Repeated concatenation
`SCAN (STRING, SET)`[a]	Scan a string for a character in a set
`TRIM (STRING)`	Remove trailing blank characters
`VERIFY (STRING, SET)`[a]	Verify the set of characters in a string
`STRING_A//STRING_B`	Concatenate two strings

[a] Optional arguments not shown.

Table B.26: F90 Character Functions.

Action	*ASCII Character*	F90 *Input*[a]	*C++ Input*
Alert (Bell)	7	Ctrl-G	\a
Backspace	8	Ctrl-H	\b
Carriage Return	13	Ctrl-M	\r
End of Transmission	4	Ctrl-D	Ctrl-D
Form Feed	12	Ctrl-L	\f
Horizontal Tab	9	Ctrl-I	\t
New Line	10	Ctrl-J	\n
Vertical Tab	11	Ctrl-K	\v

[a] "Ctrl-" denotes control action. That is, simultaneous pressing of the `CONTROL` key *and* the letter following.

Table B.27: How to type non-printing characters.

C, C++	`Variable.component.sub_component`
F90	`Variable%component%sub_component`

Table B.28: Referencing Structure Components.

C, C++	`struct data_tag {` `  intrinsic_type_1 component_names;` `  intrinsic_type_2 component_names;` `} ;`
F90	`type data_tag` `  intrinsic_type_1 :: component_names;` `  intrinsic_type_2 :: component_names;` `end type  data_tag`

Table B.29: Defining New Types of Data Structure.

C, C++	`struct data_tag {` `  intrinsic_type_1 component_names;` `  struct tag_2  component_names;` `} ;`
F90	`type data_tag` `  intrinsic_type :: component_names;` `  type (tag_2) :: component_names;` `end type data_tag`

Table B.30: Nested Data Structure Definitions.

C, C++	`struct data_tag variable_list; /* Definition */` `struct data_tag variable = {component_values}; /* Initialization */` `variable.component.sub_component = value; /* Assignment */`
F90	`type (data_tag) :: variable_list ! Definition` `variable = data_tag (component_values) ! Initialization` `variable%component%sub_component = value ! Assignment`

Table B.31: Declaring, initializing, and assigning components of user-defined datatypes.

```
INTEGER, PARAMETER ::  j_max = 6
TYPE meaning_demo
    INTEGER, PARAMETER ::  k_max = 9, word = 15
    CHARACTER (LEN = word) ::  name(k_max)
END TYPE meaning_demo
TYPE (meaning_demo) derived(j_max)
```

Construct	Interpretation
`derived`	All components of all `derived`'s elements
`derived(j)`	All components of j^{th} element of `derived`
`derived(j)%name`	All `k_max` components of `name` within j^{th} element of `derived`
`derived%name(k)`	Component `k` of the `name` array for all elements of `derived`
`derived(j)%name(k)`	Component `k` of the `name` array of j^{th} element of `derived`

Table B.32: F90 Derived Type Component Interpretation.

	C++	F90
Declaration	`type_tag *pointer_name;`	`type (type_tag), pointer :: pointer_name`
Target	`&target_name`	`type (type_tag), target ::  target_name`
Examples	`char *cp, c;` `int *ip, i;` `float *fp, f;` `cp = & c;` `ip = & i;` `fp = & f;`	`character, pointer :: cp` `integer, pointer :: ip` `real, pointer :: fp` `cp => c` `ip => i` `fp => f`

Table B.33: Definition of pointers and accessing their targets.

C, C++	`pointer_name = NULL`
F90	`nullify (list_of_pointer_names)`
F95	`pointer_name = NULL()`

Table B.34: Nullifing a Pointer to Break Association with Target.

Purpose	F90	MATLAB
Form subscripts	( )	( )
Separates subscripts & elements	,	,
Generates elements & subscripts	:	:
Separate commands	;	;
Forms arrays	(/ /)	[ ]
Continue to new line	&	...
Indicate comment	!	%
Suppress printing	default	;

Table B.35: Special Array Characters.

Description	*Equation*	*Fortran90 Operator*	*Matlab Operator*	*Original Sizes*	*Result Size*
Scalar plus scalar	$c = a \pm b$	$c = a \pm b$	$c = a \pm b;$	$1, 1$	$1, 1$
Element plus scalar	$c_{jk} = a_{jk} \pm b$	$c = a \pm b$	$c = a \pm b;$	m, n and $1, 1$	m, n
Element plus element	$c_{jk} = a_{jk} \pm b_{jk}$	$c = a \pm b$	$c = a \pm b;$	m, n and m, n	m, n
Scalar times scalar	$c = a \times b$	$c = a * b$	$c = a * b;$	$1, 1$	$1, 1$
Element times scalar	$c_{jk} = a_{jk} \times b$	$c = a * b$	$c = a * b;$	m, n and $1, 1$	m, n
Element times element	$c_{jk} = a_{jk} \times b_{jk}$	$c = a * b$	$c = a. * b;$	m, n and m, n	m, n
Scalar divide scalar	$c = a/b$	$c = a/b$	$c = a/b;$	$1, 1$	$1, 1$
Scalar divide element	$c_{jk} = a_{jk}/b$	$c = a/b$	$c = a/b;$	m, n and $1, 1$	m, n
Element divide element	$c_{jk} = a_{jk}/b_{jk}$	$c = a/b$	$c = a./b;$	m, n and m, n	m, n
Scalar power scalar	$c = a^b$	$c = a**b$	$c = a \wedge b;$	$1, 1$	$1, 1$
Element power scalar	$c_{jk} = a_{jk}^b$	$c = a**b$	$c = a \wedge b;$	m, n and $1, 1$	m, n
Element power element	$c_{jk} = a_{jk}^{b_{jk}}$	$c = a**b$	$c = a. \wedge b;$	m, n and m, n	m, n
Matrix transpose	$C_{kj} = A_{jk}$	$C = \text{transpose}\,(A)$	$C = A';$	m, n	n, m
Matrix times matrix	$C_{ij} = \sum_k A_{ik} B_{kj}$	$C = \text{matmul}(A, B)$	$C = A * B;$	m, r and r, n	m, n
Vector dot vector	$c = \sum_k A_k B_k$	$c = \text{sum}(A * B)$	$c = \text{sum}(A. * B);$	$m, 1$ and $m, 1$	$1, 1$
		$c = \text{dot\_product}(A, B)$	$c = A * B';$	$m, 1$ and $m, 1$	$1, 1$

Table B.36: Array Operations in Programming Constructs. Lower case letters denote scalars or scalar elements of arrays. Matlab arrays are allowed a maximum of two subscripts while Fortran allows seven. Upper case letters denote matrices or scalar elements of matrices.

Table B.37: Equivalent Fortran90 and MATLAB Intrinsic Functions.
The following KEY symbols are utilized to denote the TYPE of the intrinsic function, or subroutine, and its arguments: A-complex, integer, or real; I-integer; L-logical; M-mask (logical); R-real; X-real; Y-real; V-vector (rank 1 array); and Z-complex. Optional arguments are not shown. Fortran 90 and MATLAB also have very similar array operations and colon operators.

Type	Fortran90	MATLAB	Brief Description
A	ABS(A)	abs(a)	Absolute value of A.
R	ACOS(X)	acos(x)	Arc cosine function of real X.
R	AIMAG(Z)	imag(z)	Imaginary part of complex number.
R	AINT(X)	real(fix(x))	Truncate X to a real whole number.
L	ALL(M)	all(m)	True if all mask elements, M, are true.
R	ANINT(X)	real(round(x))	Real whole number nearest to X.
L	ANY(M)	any(m)	True if any mask element, M, is true.
R	ASIN(X)	asin(x)	Arcsine function of real X.
R	ATAN(X)	atan(x)	Arctangent function of real X.
R	ATAN2(Y,X)	atan2(y,x)	Arctangent for complex number(X, Y).
I	CEILING(X)	ceil(x)	Least integer >= real X.
Z	CMPLX(X,Y)	(x+yi)	Convert real(s) to complex type.
Z	CONJG(Z)	conj(z)	Conjugate of complex number Z.
R	COS(R_Z)	cos(r_z)	Cosine of real or complex argument.
R	COSH(X)	cosh(x)	Hyperbolic cosine function of real X.
I	COUNT(M)	sum(m==1)	Number of true mask, M, elements.
R,L	DOT_PRODUCT(X,Y)	x'⋆y	Dot product of vectors X and Y.
R	EPSILON(X)	eps	Number, like X, $\ll 1$.
R,Z	EXP(R_Z)	exp(r_z)	Exponential of real or complex number.
I	FLOOR(X)	floor	Greatest integer $\leq$ X.
R	HUGE(X)	realmax	Largest number like X.
I	INT(A)	fix(a)	Convert A to integer type.
R	LOG(R_Z)	log(r_z)	Logarithm of real or complex number.
R	LOG10(X)	log10(x)	Base 10 logarithm function of real X.
R	MATMUL(X,Y)	x⋆y	Conformable matrix multiplication, X*Y.
I,V	I=MAXLOC(X)	[y,i]=max(x)	Location(s) of maximum array element.
R	Y=MAXVAL(X)	y=max(x)	Value of maximum array element.
I,V	I=MINLOC(X)	[y,i]=min(x)	Location(s) of minimum array element.
R	Y=MINVAL(X)	y=min(x)	Value of minimum array element.
I	NINT(X)	round(x)	Integer nearest to real X.
A	PRODUCT(A)	prod(a)	Product of array elements.
call	RANDOM_NUMBER(X)	x=rand	Pseudo-random numbers in $(0, 1)$.
call	RANDOM_SEED	rand('seed')	Initialize random number generator.
R	REAL (A)	real(a)	Convert A to real type.
R	RESHAPE(X, (/ I, I2 /))	reshape(x, i, i2)	Reshape array X into I$\times$I2 array.
I,V	SHAPE(X)	size(x)	Array (or scalar) shape vector.
R	SIGN(X,Y)		Absolute value of X times sign of Y.
R	SIGN(0.5,X)-SIGN(0.5,-X)	sign(x)	Signum, normalized sign, –1, 0, or 1.
R,Z	SIN(R_Z)	sin(r_z)	Sine of real or complex number.
R	SINH(X)	sinh(x)	Hyperbolic sine function of real X.
I	SIZE(X)	length(x)	Total number of elements in array X.
R,Z	SQRT(R_Z)	sqrt(r_z)	Square root, of real or complex number.
R	SUM(X)	sum(x)	Sum of array elements.

(continued)

Type	Fortran90	MATLAB	Brief Description
R	TAN(X)	tan(x)	Tangent function of real X.
R	TANH(X)	tanh(x)	Hyperbolic tangent function of real X.
R	TINY(X)	realmin	Smallest positive number like X.
R	TRANSPOSE(X)	x'	Matrix transpose of any type matrix.
R	X=1	x=ones(length(x))	Set all elements to 1.
R	X=0	x=zero(length(x))	Set all elements to 0.

For more detailed descriptions and example uses of these intrinsic functions see Adams, J.C., *et al.*, *Fortran 90 Handbook*, McGraw-Hill, New York, 1992, ISBN 0–07–000406–4.

B.2 Alphabetical Table of Fortran 90 Intrinsic Routines

The following KEY symbols are utilized to denote the TYPE of the intrinsic function, or subroutine, and its arguments: A-complex, integer, or real; B-integer bit; C-character; D-dimension; I-integer; K-kind; L-logical; M-mask (logical); N-integer, or real; P-pointer; R-real; S-string; T-target; V-vector (rank one array); X-real; Y-real; Z-complex; and *-any type. For more detailed descriptions and example uses of these intrinsic functions see Adams, J.C., et al., *Fortran 90 Handbook*, McGraw-Hill, New York, 1992, ISBN 0–07–000406–4.

C++	–	int	–	–	floor	ceil
F90	aint	int	anint	nint	floor	ceiling
MATLAB	real(fix)	fix	real(round)	round	floor	ceil
Argument	**Value of Result**					
–2.000	–2.0	–2	–2.0	–2	–2	–2
–1.999	–1.0	–1	–2.0	–2	–2	–1
–1.500	–1.0	–1	–2.0	–2	–2	–1
–1.499	–1.0	–1	–1.0	–1	–2	–1
–1.000	–1.0	–1	–1.0	–1	–1	–1
–0.999	0.0	0	–1.0	–1	–1	0
–0.500	0.0	0	–1.0	–1	–1	0
–0.499	0.0	0	0.0	0	–1	0
0.000	0.0	0	0.0	0	0	0
0.499	0.0	0	0.0	0	0	1
0.500	0.0	0	1.0	1	0	1
0.999	0.0	0	1.0	1	0	1
1.000	1.0	1	1.0	1	1	1
1.499	1.0	1	1.0	1	1	2
1.500	1.0	1	2.0	2	1	2
1.999	1.0	1	2.0	2	1	2
2.000	2.0	2	2.0	2	2	2

Table B.38: Truncating Numbers.

```
WHERE (logical_array_expression)
    true_array_assignments
ELSEWHERE
    false_array_assignments
END WHERE
```

```
WHERE (logical_array_expression) true_array_assignment
```

Table B.39: F90 `WHERE` Constructs.

Function	*Description*	*Opt*	*Example*
`all`	Find if all values are true, for a fixed dimension.	d	`all(B = A, DIM = 1)` (true, false, false)
`any`	Find if any value is true, for a fixed dimension.	d	`any (B > 2, DIM = 1)` (false, true, true)
`count`	Count number of true elements for a fixed dimension.	d	`count(A = B, DIM = 2)` (1, 2)
`maxloc`	Locate first element with maximum value given by mask.	m	`maxloc(A, A < 9)` (2, 3)
`maxval`	Max element, for fixed dimension, given by mask.	b	`maxval (B, DIM=1, B > 0)` (2, 4, 6)
`merge`	Pick true array, A, or false array, B, according to mask, L.	–	`merge(A, B, L)` $\begin{bmatrix} 0 & 3 & 5 \\ 2 & 4 & 8 \end{bmatrix}$
`minloc`	Locate first element with minimum value given by mask.	m	`minloc(A, A > 3)` (2, 2)
`minval`	Min element, for fixed dimension, given by mask.	b	`minval(B, DIM = 2)` (1, 2)
`pack`	Pack array, A, into a vector under control of mask.	v	`pack(A, B < 4)` (0, 7, 3)
`product`	Product of all elements, for fixed dimension, controlled by mask.	b	`product(B)` ; (720) `product(B, DIM = 1, T)` (2, 12, 30)
`sum`	Sum all elements, for fixed dimension, controlled by mask.	b	`sum(B)` ;(21) `sum(B, DIM = 2, T)` (9, 12)
`unpack`	Replace the true locations in array B controlled by mask L with elements from the vector U.	–	`unpack(U, L, B)` $\begin{bmatrix} 7 & 3 & 8 \\ 2 & 4 & 9 \end{bmatrix}$

$$A = \begin{bmatrix} 0 & 3 & 5 \\ 7 & 4 & 8 \end{bmatrix}, \quad B = \begin{bmatrix} 1 & 3 & 5 \\ 2 & 4 & 6 \end{bmatrix}, \quad L = \begin{bmatrix} T & F & T \\ F & F & T \end{bmatrix}, \quad U = (7, 8, 9)$$

Table B.40: F90 Array Operators with Logic Mask Control. T and F denote true and false, respectively. Optional arguments: `b -- DIM & MASK`, `d -- DIM`, `m -- MASK`, `v -- VECTOR` and `DIM = 1` implies for any rows, `DIM = 2` for any columns, and `DIM = 3` for any plane.

Type	**Intrinsic**	**Description**
A	ABS (A)	Absolute value of A.
C	ACHAR (I)	Character in position I of ASCII collating sequence.
R	ACOS (X)	Arc cosine (inverse cosine) function of real X.
C	ADJUSTL (S)	Adjust S left, move leading blanks to trailing blanks.
C	ADJUSTR (S)	Adjust S right, move trailing blanks to leading blanks.
R	AIMAG (Z)	Imaginary part of complex number, Z.
R	AINT (X [,K])	Truncate X to a real whole number, of the given kind.
L	ALL (M [,D])	True if all mask, M, elements are true, in dimension D.
L	ALLOCATED (*_ARRAY_P)	True if the array or pointer is allocated.

(continued)

Alphabetic Table of Fortran90 Intrinsic Functions (continued)		
Type	**Intrinsic**	**Description**
R	ANINT (X [,K])	Real whole number nearest to X, of the given kind.
L	ANY (M [,D])	True if any mask, M, element is true, in dimension D.
R	ASIN (X)	Arcsine (inverse sine) function of real X.
L	ASSOCIATED (P [,T])	True if pointer, P, is associated with any target, or T.
R	ATAN (X)	Arctangent (inverse tangent) function of real X.
R	ATAN2 (Y,X)	Arctangent for argument of complex number (X, Y).
I	BIT_SIZE (I)	Maximum number of bits integer I can hold, e.g. 32.
L	BTEST (I,I_POS)	True if bit location I_POS of integer I has value 1.
I	CEILING (X)	Least integer $\geq$ real X, of the given kind.
C	CHAR (I [,K])	Character in position I of processor collating sequence.
Z	CMPLX (X [,Y][,K])	Convert real(s) to complex type, of given kind.
Z	CONJG (Z)	Conjugate of complex number Z.
R	COS (R_Z)	Cosine function of real or complex argument.
R	COSH (X)	Hyperbolic cosine function of real X.
I	COUNT (M [,D])	Number of true mask, M, elements, in dimension D.
$\star$	CSHIFT ($\star$_ARAY,I_SHIF [,D])	Circular shift out and in for I_SHIF elements.
call	DATE_AND_TIME ([S_DATE] [,S_TIME] [,S_ZONE] [,I_V_VALUES])	Real-time clock date, time, zone, and vector with year, month, day, UTC, hour, minutes, seconds, and milliseconds.
R	DBLE (A)	Convert A to double precision real.
N	DIGITS (N)	Number of significant digits for N, e.g. 31.
R	DIM (X,Y)	The difference, MAX (X – Y, 0.0).
N,L	DOT_PRODUCT (V,V_2)	Dot product of vectors V and V_2.
R	DPROD (X,Y)	Double precision real product of two real scalars.
$\star$	EOSHIFT ($\star$_ARRAY, I_SHIFT [,$\star$_FILL][,D])	Perform vector end-off shift by $\pm$ I_shift terms, and fill, in dimension D.
R	EPSILON (X)	Number $\ll 1$, for numbers like X, e.g. 2$\star\star$–23.
R,Z	EXP (R_Z)	Exponential function of real or complex argument.
I	EXPONENT (X)	Exponent part of the model for real X.
I	FLOOR (X)	Greatest integer less than or equal to X.
R	FRACTION (X)	Fractional part of the model for real X.
N	HUGE (N)	Largest number for numbers like N, e.g. 2$\star\star$128.
I	IACHAR (C)	Position of character C in ASCII collation.
B	IAND (I,I_2)	Logical AND on the bits of I and I_2.
B	IBCLR (I,I_POS)	Clear bit I_POS to zero in integer I.
B	IBITS (I,I_POS,I_LEN)	Extract an I_LEN sequence of bits at I_POS in I.
B	IBSET (I,I_POS)	Set bit I_POS to one in integer I.
I	ICHAR (C)	Position of character C in processor collation.
B	IEOR (I,I_2)	Exclusive OR on the bits of I and I_2.
I	INDEX (S,S_SUB [,L_BACK])	Left starting position of S_SUB within S (right).
I	INT (A [,K])	Convert A to integer type, of given kind.
B	IOR (I,I_2)	Inclusive OR on the bits of I and I_2.
B	ISHFT (I,I_SHIFT)	Logical shift of bits of I by I_SHIFT, pad with 0.
B	ISHFTC (I,I_SHIFT [,I_SIZE])	Logical circular shift of I_SIZE rightmost bits of I.
I	KIND (ANY)	Kind type integer parameter value for any argument.
I,V	LBOUND ($\star$_ARRAY [,D])	ARRAY lower bound(s) vector, along dimension D.
I	LEN (S)	Total character string length.
I	LEN_TRIM (S)	Length of S without trailing blanks.
L	LGE (S,S_2)	True if S $>$ or equal to S_2 in ASCII sequence.
L	LGT (S,S_2)	True if S follows S_2 in ASCII collating sequence.

(continued)

Alphabetic Table of Fortran90 Intrinsic Functions (continued)		
Type	**Intrinsic**	**Description**
L	LLE (S,S_2)	True if S $<$ or equal to S_2 in ASCII sequence.
L	LLT (S,S_2	True if S precedes S_2 in ASCII collating sequence.
R	LOG (R_Z)	Natural (base e) logarithm of real or complex number.
L	LOGICAL (L [,K])	Convert L to logical of kind K.
R	LOG10 (X)	Common (base 10) logarithm function of real X.
N,L	MATMUL (MATRIX,MATRIX_2)	Conformable matrix multiplication.
N	MAX (N,N_2 [,N_3,...])	Maximum value of two or more numbers same type.
I	MAXEXPONENT (X)	Maximum exponent for real numbers like X, e.g. 128.
I,V	MAXLOC (N_ARRAY [,M])	Location(s) of maximum ARRAY element, passing M.
N	MAXVAL (N_ARRAY [,D] [,M])	Maximum ARRAY term, in dimension D, passing M.
$\star$	MERGE ($\star$_TRUE,$\star$_FALSE,M)	Use $\star$_TRUE when M is true; $\star$_FALSE otherwise.
N	MIN (N,N_2 [,N_3,...])	Minimum value of two or more same type numbers.
I	MINEXPONENT (X)	Minimum exponent for real numbers like X, e.g. –125.
I,V	MINLOC (N_ARRAY [,M])	Location(s) of minimum ARRAY term, passing M.
N	MINVAL (N_ARRAY [,D] [,M])	Minimum ARRAY term, in dimension D, passing M.
N	MOD (N,N_2)	Remainder for N_2. That is, N–INT(N/N_2)$\star$N_2.
N	MODULO (N,N_2)	Modulo, that is, N–FLOOR(N/N_2)$\star$N_2.
call	MVBITS (I_FROM,I_LOC, I_LEN,I_TO,I_POS)	Copy I_LEN bits at I_LOC in I_FROM to I_TO at I_POS.
R	NEAREST (X,Y)	Nearest number at X in the direction of sign Y.
I	NINT (X [,K])	Integer nearest to real X, of the stated kind.
I	NOT (I)	Logical complement of the bits of integer I.
$\star$,V	PACK ($\star$_ARRAY,M [,V_PAD])	Pack ARRAY at true M into vector, using V_PAD.
I	PRECISION (R_Z)	Decimal precision for a real or complex R_Z, e.g. 6.
L	PRESENT (OPTIONAL)	True if optional argument is present in call.
A	PRODUCT (A_ARRAY [,D] [,M])	Product of ARRAY elements, along D, for mask M.
I	RADIX (N)	Base of the model for numbers like N, e.g. 2.
call	RANDOM_NUMBER (X)	Pseudo-random numbers in range $0 < X < 1$.
call	RANDOM_SEED ([I_SIZE] [,I_V_PUT][,I_V_GET])	Initialize random number generator, defaults to processor initialization.
I	RANGE (A)	Decimal exponent range in the model for A, e.g. 37.
R	REAL (A [,K])	Convert A to real type, of type K.
S	REPEAT (S,I_COPIES)	Concatenates I_COPIES of string S.
$\star$	RESHAPE ($\star$_ARAY,I_V_SHAP [,$\star$_PAD] [,V_ORDER])	Reshape ARAY, using vector SHAP, pad from an array, and re-order.
R	RRSPACING (X)	Relative spacing reciprocal of numbers near X.
R	SCALE (X,I)	Return X times b$\star\star$I, for base of b = RADIX (X).
I	SCAN (S,S_SET [,L_BACK])	Leftmost character index in S found in S_SET; (right-most).
I	SELECTED_INT_KIND (I_r)	Integer kind with range, –(10$\star\star$I_r) to (10$\star\star$I_r).
I	SELECTED_REAL_KIND ([I] [,I_r])	Kind for real of decimal precision, I, and exponent range, I_r.
R	SET_EXPONENT (X,I)	Number with mantissa of X and exponent of I.
I,V	SHAPE ($\star$_ARRAY)	ARRAY (or scalar) shape vector.
N	SIGN (N,N_2)	Absolute value of N times sign of same type N_2.
R,Z	SIN (R_Z)	Sine function of real or complex number.
R	SINH (X)	Hyperbolic sine function of real X.
I	SIZE ($\star$_ARRAY [,D])	ARRAY size, along dimension D.
R	SPACING (X)	Absolute spacing of numbers near real X, e.g. 2$\star\star$–17.
$\star$	SPREAD ($\star$_ARAY,D,I_COPIES)	I_COPIES along dimension D of ARAY into an array of rank 1 greater.

(continued)

Alphabetic Table of Fortran90 Intrinsic Functions (continued)		
Type	**Intrinsic**	**Description**
R,Z	SQRT (R_Z)	Square root function, of real or complex number.
A	SUM (A_ARRAY [,D] [,M])	Sum of ARRAY elements, along D, passing mask M.
call	SYSTEM_CLOCK ([I_NOW] [,I_RATE] [,I_MAX])	Integer data from real-time clock. CPU time is (finish_now - start_now) / rate.
R	TAN (X)	Tangent function of real X.
R	TANH (X)	Hyperbolic tangent function of real X.
R	TINY (N)	Smallest positive number, like N, e.g. 2⋆⋆–126.
⋆	TRANSFER (⋆_ARAY, V_MOLD [,I_SIZE])	Same representation as ARAY, but type of MOLD, in vector of length SIZE.
⋆	TRANSPOSE (MATRIX)	Matrix transpose of any type matrix.
S	TRIM (S)	Remove trailing blanks from a single string.
I,V	UBOUND (⋆_ARRAY [,D])	ARRAY upper bound(s) vector, along dimension D.
⋆	UNPACK (V,M,⋆_USE)	Unpack vector V at true elements of M, into USE.
I	VERIFY (S,S_SET [,L_BACK])	First position in S not found in S_SET (or last).

Subject Table of Fortran 90 Intrinsic Routines

The following KEY symbols are utilized to denote the TYPE of the intrinsic function, or subroutine, and its arguments: A-complex, integer, or real; B-integer bit; C-character; D-dimension; I-integer; K-kind; L-logical; M-mask (logical); N-integer, or real; P-pointer; R-real; S-string; T-target; V-vector (rank one array); X-real; Y-real; Z-complex; and *-any type. For more detailed descriptions and example uses of these intrinsic functions see Adams, J.C., et al., *Fortran 90 Handbook*, McGraw-Hill, New York, 1992, ISBN 0–07–000406–4.

Type	**Intrinsic**	**Description**
ALLOCATION		
L	ALLOCATED (⋆_ARRAY)	True if the array is allocated.
ARGUMENT		
L	PRESENT (OPTIONAL)	True if optional argument is present in the call.
ARRAY: CONSTRUCTION		
⋆	MERGE (⋆_TRUE,⋆_FALSE,M)	Use ⋆_TRUE if M is true; ⋆_FALSE otherwise.
⋆,V	PACK (⋆_ARRAY,M [,V_PAD])	Pack ARRAY for true M into vector, pad from V_PAD.
⋆	RESHAPE (⋆_ARRAY,I_V_SHAPE [,⋆_PAD] [,V_ORDER])	Reshape ARRAY using vector SHAPE, pad from an array, and re-order.
⋆	SPREAD (⋆_ARRAY,D,I_COPIES)	I_COPIES along D of ARRAY to rank 1 greater array.
⋆	UNPACK (V,M,⋆_USE)	Unpack V at true elements of M, into USE.
ARRAY: DIMENSIONS		
I,V	LBOUND (⋆_ARRAY [,D])	ARRAY lower bound(s vector), along dimension D.
I,V	SHAPE (⋆_ARRAY)	ARRAY (or scalar) shape vector.
I	SIZE (⋆_ARRAY [,D])	ARRAY size, along dimension D.
I,V	UBOUND (⋆_ARRAY [,D])	ARRAY upper bound(s vector), along dimension D.
ARRAY: INQUIRY		
L	ALL (M [,D])	True if all mask, M, elements are true, along D.
L	ALLOCATED (⋆_ARRAY)	True if the array is allocated.
L	ANY (M [,D])	True if any mask, M, element is true, along D.
I,V	LBOUND (⋆_ARRAY [,D])	ARRAY lower bound(s) vector, along dimension D.
I,V	SHAPE (⋆_ARRAY)	ARRAY (or scalar) shape vector.

(continued)

Subject Table of Fortran 90 Intrinsic Functions (continued)		
Type	**Intrinsic**	**Description**
I,V	UBOUND (⋆_ARRAY [,D])	ARRAY upper bound(s) vector, along dimension D.
ARRAY: LOCATION		
I,V	MAXLOC (N_ARRAY [,M])	Location(s) of maximum ARRAY term, passing M.
I,V	MINLOC (N_ARRAY [,M])	Location(s) of minimum ARRAY term, passing M.
ARRAY: MANIPULATION		
⋆	CSHIFT (⋆_ARRAY,I_SHIFT [,D])	Circular shift out and in for I_SHIFT elements.
⋆	EOSHIFT (⋆_ARRAY,I_SHIFT [,⋆_FIL][,D])	End-off shift ARRAY, and fill, in dimension D.
⋆	TRANSPOSE (MATRIX)	Matrix transpose of any type matrix.
ARRAY: MATHEMATICS		
N,L	DOT_PRODUCT (V,V_2)	Dot product of vectors V and V_2.
N,L	MATMUL (MATRIX,MATRIX_2)	Conformable matrix multiplication.
N	MAXVAL (N_ARRAY [,D] [,M])	Value of max ARRAY term, along D, passing M.
N	MINVAL (N_ARRAY [,D] [,M])	Value of min ARRAY term, along D, passing M.
A	PRODUCT (A_ARRAY [,D] [,M])	Product of ARRAY terms, along D, for mask M.
A	SUM (A_ARRAY [,D] [,M])	Sum of ARRAY terms, along D, passing mask M.
ARRAY: PACKING		
⋆,V	PACK (⋆_ARRAY,M [,V_PAD])	Pack ARRAY for true M into vector, pad from V_PAD.
⋆	UNPACK (V,M,⋆_USE)	Unpack V at true elements of M, into USE.
ARRAY: REDUCTION		
L	ALL (M [,D])	True if all mask, M, terms are true, along D.
L	ANY (M [,D])	True if any mask, M, term is true, along D.
I	COUNT (M [,D])	Number of true mask, M, terms, along dimension D.
N	MAXVAL (N_ARRAY [,D] [,M])	Value of max ARRAY term, along D, passing M.
N	MINVAL (N_ARRAY [,D] [,M])	Value of min ARRAY term, along D, passing M.
A	PRODUCT (A_ARRAY [,D] [,M])	Product of ARRAY terms, along D, for mask M.
A	SUM (A_ARRAY [,D] [,M])	Sum of ARRAY terms, along D, passing mask M.
BACK SCAN		
I	INDEX (S,S_SUB [,L_BACK])	Left starting position of S_SUB within S (or right).
I	SCAN (S,S_SET [,L_BACK])	Left character index in S also in S_SET (or right).
I	VERIFY (S,S_SET [,L_BACK])	First position in S not belonging to S_SET (or last).
BIT: INQUIRY		
I	BIT_SIZE (I)	Max number of bits possible in integer I, e.g. 32.
BIT: MANIPULATION		
L	BTEST (I,I_POS)	True if bit location I_POS of integer I has value one.
B	IAND (I,I_2)	Logical AND on the bits of I and I_2.
B	IBCLR (I,I_POS)	Clear bit I_POS to zero in integer I.
B	IBITS (I,I_POS,I_LEN)	Extract I_LEN bits at I_POS in integer I.
B	IBSET (I,I_POS)	Set bit I_POS to one in integer I.
B	IEOR (I,I_2)	Exclusive OR on the bits of I and I_2.
B	IOR (I,I_2)	Inclusive OR on the bits of I and I_2.
B	ISHFT (I,I_SHIFT)	Logical shift of bits of I by I_SHIFT, pad with 0.
B	ISHFTC (I,I_SHIFT [,I_SIZE])	Logical circular shift of I_SIZE rightmost bits of I.
call	MVBITS (I_GET, I_LOC, I, I_TO,I_POS)	Copy I bits at I_LOC in I_GET to I_TO at I_POS.
I	NOT (I)	Logical complement of the bits of integer I.
⋆	TRANSFER (⋆_ARRAY,	

(continued)

Subject Table of Fortran 90 Intrinsic Functions (continued)		
Type	**Intrinsic**	**Description**
	V_MOLD [,I_SIZE])	Same representation as ARRAY, but type of MOLD.
BOUNDS		
I	CEILING (X)	Least integer greater than or equal to real X.
I	FLOOR (X)	Greatest integer less than or equal to X.
I,V	LBOUND (⋆_ARRAY [,D])	ARRAY lower bound(s) vector, along dimension D.
N	MAX (N,N_2 [,N_3,...])	Maximum value of two or more numbers same type.
N	MAXVAL (N_ARRAY [,D] [,M])	Value of max ARRAY term, along D, passing M.
N	MINVAL (N_ARRAY [,D] [,M])	Value of min ARRAY term, along D, passing M.
I,V	UBOUND (⋆_ARRAY [,D])	ARRAY upper bound(s) vector, along dimension D.
CALLS		
call	MVBITS (I_GET,I_LOC,I, I_TO,I_POS)	Copy I bits at I_LOC in I_GET to I_TO at I_POS.
call	DATE_AND_TIME ([S_DATE] [,S_TIME] [,S_ZONE] [,I_V_VALUES])	Real-time clock data.
call	RANDOM_NUMBER (X)	Pseudo-random numbers in range $0 < X < 1$.
call	RANDOM_SEED ([I_SIZE] [,I_V_P] [,I_V_G])	Initialize random number generator.
call	SYSTEM_CLOCK ([I_NOW] [,I_RAT] [,I_MX])	Integer data from real-time clock.
CHARACTERS		
C	ACHAR (I)	Character in position I of ASCII collating sequence.
C	CHAR (I [,K])	Character in position I of processor collation.
I	IACHAR (C)	Position of character C in ASCII collating sequence.
I	ICHAR (C)	Position of character C in processor collation.
CLOCK		
call	SYSTEM_CLOCK ([I_NOW] [,I_RAT] [,I_MX])	Integer data from real-time clock.
COMBINING		
⋆	MERGE (⋆_TRUE,⋆_FALSE,M)	Use ⋆_TRUE term if M is true or ⋆_FALSE otherwise.
COMPLEX		
R	AIMAG (Z)	Imaginary part of complex number.
Z	CMPLX (X [,Y][,K])	Convert real(s) to complex type, of given kind.
Z	CONJG (Z)	Conjugate of complex number Z.
R	COS (R_Z)	Cosine function of real or complex argument.
R,Z	EXP (R_Z)	Exponential function of real or complex argument.
R	LOG (R_Z)	Natural (base e) logarithm of real or complex number.
I	PRECISION (R_Z)	Decimal precision of real or complex value, e.g. 6.
R,Z	SIN (R_Z)	Sine function of real or complex number.
R,Z	SQRT (R_Z)	Square root function, of real or complex number.
CONVERSIONS		
R	AIMAG (Z)	Imaginary part of complex number.
R	AINT (X [,K])	Truncate X to a real whole number.
Z	CMPLX (X [,Y][,K])	Convert real (s) to complex type, of given kind.
R	DBLE (A)	Convert A to double precision real.
R	DPROD (X,Y)	Double precision product of two default real scalars.

(continued)

Subject Table of Fortran 90 Intrinsic Functions (continued)		
Type	**Intrinsic**	**Description**
I	INT (A [,K])	Convert A to integer type, of given kind.
L	LOGICAL (L [,K])	Convert L to logical of kind K.
I	NINT (X [,K])	Integer nearest to real X, of the stated kind.
R	REAL (A [,K])	Convert A to real type, of type K.
N	SIGN (N,N_2)	Absolute value of N times sign of same type N_2.
⋆	TRANSFER (⋆_ARRAY, V_MOLD [,I_SIZ])	Same representation as ARRAY, but type of MOLD.
COPIES		
⋆	MERGE (⋆_TRUE,⋆_FALSE,M)	Use ⋆_TRUE if M is true or ⋆_FALSE otherwise.
call	MVBITS (I_FROM,I_LOC, I, I_TO,I_POS)	Copy I bits at I_LOC in I_FROM to I_TO at I_POS.
S	REPEAT (S,I_COPIES)	Concatenates I_COPIES of string S.
⋆	SPREAD (⋆_ARRAY,D,I_COPIES)	I_COPIES along D of ARRAY to rank 1 greater array.
COUNTING		
I	COUNT (M [,D])	Number of true mask, M, terms, along dimension D.
DATE		
call	DATE_AND_TIME ([S_DATE] [,S_TIME] [,S_ZONE] [,I_V_VALUES])	Real-time clock data.
DIMENSION OPTIONAL ARGUMENT		
L	ALL (M [,D])	True if all mask, M, terms are true, along D.
L	ANY (M [,D])	True if any mask, M, term is true, along D.
I	COUNT (M [,D])	Number of true mask, M, terms, along dimension D.
⋆	CSHIFT (⋆_ARRAY,I_SHIFT [,D])	Perform circular shift out and in for I_SHIFT terms.
⋆	EOSHIFT (⋆_ARRAY, I_SHIFT [,⋆_FIL][,D])	Perform end-off shift, and fill, in dimension D.
I,V	LBOUND (⋆_ARRAY [,D])	ARRAY lower bound(s) vector, along dimension D.
N	MAXVAL (N_ARRAY [,D] [,M])	Value of max ARRAY term, along D, passing M.
N	MINVAL (N_ARRAY [,D] [,M])	Value of min ARRAY term, along D, passing M.
A	PRODUCT (A_ARRAY [,D] [,M])	Product of ARRAY terms, along D, for mask M.
I	SIZE (⋆_ARRAY [,D])	ARRAY size, along dimension D.
A	SUM (A_ARRAY [,D] [,M])	Sum of ARRAY terms, along D, passing mask M.
I,V	UBOUND (⋆_ARRAY [,D])	ARRAY upper bound(s) vector, along dimension D.
DIMENSIONS		
I,V	LBOUND (⋆_ARRAY [,D])	ARRAY lower bound(s) vector, along dimension D.
I,V	SHAPE (⋆_ARRAY)	ARRAY (or scalar) shape vector.
I	SIZE (⋆_ARRAY [,D])	ARRAY size, along dimension D.
I,V	UBOUND (⋆_ARRAY [,D])	ARRAY upper bound(s) vector, along dimension D.
DOUBLE PRECISION		(see SELECTED_REAL_KIND)
R	DBLE (A)	Convert A to double precision real.
R	DPROD (X,Y)	Double precision product of two default real scalars.
EXISTENCE		
L	ALLOCATED (⋆_ARRAY)	True if the array is allocated.
L	ASSOCIATED (P [,T])	True if pointer, P, is associated with any target, or T.
L	PRESENT (OPTIONAL)	True if optional argument is present in call.
FILE		

(continued)

Subject Table of Fortran 90 Intrinsic Functions (continued)		
Type	**Intrinsic**	**Description**
FILL IN		
⋆	EOSHIFT (⋆_ARRAY,I_SHIFT [,⋆_FIL][,D])	End-off shift ARRAY, and fill, in dimension D.
INQUIRY: ARRAY		
L	ALL (M [,D])	True if all mask, M, terms are true, along D.
L	ALLOCATED (⋆_ARRAY)	True if the array is allocated.
L	ANY (M [,D])	True if any mask, M, term is true, along D.
I,V	LBOUND (⋆_ARRAY [,D])	ARRAY lower bound(s) vector, along dimension D.
I,V	SHAPE (⋆_ARRAY)	ARRAY (or scalar) shape vector.
I	SIZE (⋆_ARRAY [,D])	ARRAY size, along dimension D.
I,V	UBOUND (⋆_ARRAY [,D])	ARRAY upper bound(s) vector, along dimension D.
INQUIRY: BIT		
I	BIT_SIZE (I)	Max number of bits possible in integer I, e.g. 32.
INQUIRY: CHARACTER		
I	LEN (S)	Total character string length.
I	LEN_TRIM (S)	Length of S without trailing blanks.
INQUIRY: NUMBER MODEL		
N	DIGITS (N)	Number of significant digits in number N, e.g. 31.
R	EPSILON (X)	Number $\ll 1$, for numbers like X, e.g. 2⋆⋆–23.
N	HUGE (N)	Largest number for numbers like N, e.g. 2⋆⋆128.
I	MAXEXPONENT (X)	Max exponent for real numbers like X, e.g. 128.
I	MINEXPONENT (X)	Min exponent for real numbers like X, e.g. –125.
I	PRECISION (R_Z)	Decimal precision for real or complex value, e.g. 6.
I	RADIX (N)	Base of the model for numbers like N, e.g. 2.
I	RANGE (A)	Decimal exponent range for A, e.g. 37.
I,V	SHAPE (⋆_ARRAY)	ARRAY (or scalar) shape vector.
I	SIZE (⋆_ARRAY [,D])	ARRAY size, along dimension D.
R	TINY (N)	Smallest positive number, like N, e.g. 2⋆⋆–126.
INQUIRY: MISCELLANEOUS		
I	COUNT (M [,D])	Number of true mask, M, elements, along D.
I	INDEX (S,S_SUB [,L_BACK])	Left starting position of S_SUB within S (or right).
I	SCAN (S,S_SET [,L_BACK])	Left character index in S also in S_SET; (or right).
I	VERIFY (S,S_SET [,L_BACK])	First position in S not belonging to S_SET, (or last).
INTEGERS		
I	CEILING (X)	Least integer greater than or equal to real X.
I	FLOOR (X)	Greatest integer less than or equal to X.
I	MAX1 (X,X2 [,X3])	Maximum integer from list of reals
I	MIN1 (X,X2 [,X3])	Minimum integer from list of reals
N	MODULO (N,N_2)	Modulo, N-FLOOR(N/N_2)⋆N_2.
I	SELECTED_INT_KIND (I_r)	Integer with exponent, –(10⋆⋆I_r) to (10⋆⋆I_r).
KIND: INQUIRY		
I	KIND (ANY)	Kind type integer parameter value for any argument.
KIND: DEFINITION		
I	SELECTED_INT_KIND (I_r)	Integer with exponent, –(10⋆⋆I_r) to (10⋆⋆I_r).
I	SELECTED_REAL_KIND ([I] [,I_r])	Real with precision, I, and exponent range, I_r.
KIND: USE OPTION		

(continued)

Subject Table of Fortran 90 Intrinsic Functions (continued)		
Type	**Intrinsic**	**Description**
R	AINT (X [,K])	Truncate X to a real whole number.
R	ANINT (X [,K])	Real whole number nearest to X.
C	CHAR (I [,K])	Character in position I of processor collation.
Z	CMPLX (X [,Y][,K])	Convert real(s) to complex type, of given kind.
I	INT (A [,K])	Convert A to integer type, of given kind.
L	LOGICAL (L [,K])	Convert L to logical of kind K.
I	NINT (X [,K])	Integer nearest to real X, of the stated kind.
R	REAL (A [,K])	Convert A to real type, of type K.
LOCATION		
I	IACHAR (C)	Position of character C in ASCII collating sequence.
I	ICHAR (C)	Position of character C in processor collation.
I	INDEX (S,S_SUB [,L_BACK])	Left starting position of S_SUB within S (or right).
I,V	MAXLOC (N_ARRAY [,M])	Vector location(s) of ARRAY maximum, passing M.
I,V	MINLOC (N_ARRAY [,M])	Vector location(s) of ARRAY minimum, passing M.
I	SCAN (S,S_SET [,L_BACK])	Left character index in S found in S_SET; (or right).
LOGICAL		
L	ALL (M [,D])	True if all mask, M, terms are true, along D.
L	ALLOCATED ($\star$_ARRAY)	True if the array is allocated.
L	ANY (M [,D])	True if any mask, M, term is true, along D.
L	ASSOCIATED (P [,T])	True if pointer, P, is associated with any target, or T.
L	BTEST (I,I_POS)	True if bit location I_POS of integer I has value one.
N,L	DOT_PRODUCT (V,V_2)	Dot product of vectors V and V_2.
B	IAND (I,I_2)	Logical AND on the bits of I and I_2.
B	IEOR (I,I_2)	Exclusive OR on the bits of I and I_2.
B	IOR (I,I_2)	Inclusive OR on the bits of I and I_2.
B	ISHFT (I,I_SHIFT)	Logical shift of bits of I by I_SHIFT, pad with 0.
L	LGE (S,S_2)	True if S is $\geq$ S_2 in ASCII collating sequence.
L	LGT (S,S_2)	True if S follows S_2 in ASCII collating sequence.
L	LLE (S,S_2)	True if S is $\leq$ to S_2 in ASCII collating sequence.
L	LLT (S,S_2)	True if S precedes S_2 in ASCII collating sequence.
N,L	MATMUL (MATRIX,MATRIX_2)	Conformable matrix multiplication.
L	LOGICAL (L [,K])	Convert L to logical of kind K.
I	NOT (I)	Logical complement of the bits of integer I.
L	PRESENT (OPTIONAL)	True if optional argument is present in call.
MASK, or MASK OPTIONAL ARGUMENT		
L	ALL (M [,D])	True if all mask, M, terms are true, along D.
L	ANY (M [,D])	True if any mask, M, term is true, along D.
I	COUNT (M [,D])	Number of true mask, M, terms, along dimension D.
I,V	MAXLOC (N_ARRAY [,M])	Vector of location(s) of ARRAY max's, passing M.
N	MAXVAL (N_ARRAY [,D] [,M])	Value of ARRAY maximum, along D, passing M.
$\star$	MERGE ($\star$_TRUE,$\star$_FALSE,M)	Use $\star$_TRUE if M is true or $\star$_FALSE otherwise.
I,V	MINLOC (N_ARRAY [,M])	Vector location(s) of ARRAY minimum, passing M.
N	MINVAL (N_ARRAY [,D] [,M])	Value of ARRAY minimum, along D, passing M.
$\star$,V	PACK ($\star$_ARRAY,M [,V_PAD])	Pack ARRAY for true M into vector, pad from V_PAD.
A	PRODUCT (A_ARRAY [,D] [,M])	Product of ARRAY terms, along D, for mask M.
A	SUM (A_ARRAY [,D] [,M])	Sum of ARRAY terms, along D, passing mask M.
MATHEMATICAL FUNCTIONS		
R	ACOS (X)	Arc cosine (inverse cosine) function of real X.

(continued)

Subject Table of Fortran 90 Intrinsic Functions (continued)		
Type	**Intrinsic**	**Description**
R	ASIN (X)	Arcsine (inverse sine) function of real X.
R	ATAN (X)	Arctangent (inverse tangent) function of real X.
R	ATAN2 (Y,X)	Arctangent for argument of complex number (X, Y).
R	COS (R_Z)	Cosine function of real or complex argument.
R	COSH (X)	Hyperbolic cosine function of real X.
R,Z	EXP (R_Z)	Exponential function of real or complex argument.
R	LOG (R_Z)	Natural logarithm of real or complex number.
R	LOG10 (X)	Common (base 10) logarithm function of real X.
R,Z	SIN (R_Z)	Sine function of real or complex number.
R	SINH (X)	Hyperbolic sine function of real X.
R	TAN (X)	Tangent function of real X.
R	TANH (X)	Hyperbolic tangent function of real X.
MATRICES (See ARRAYS)		
N,L	DOT_PRODUCT (V,V_2)	Dot product of vectors V and V_2.
N,L	MATMUL (MATRIX,MATRIX_2)	Conformable matrix multiplication.
⋆	TRANSPOSE (MATRIX)	Matrix transpose of any type matrix.
NUMBER MODEL		
N	DIGITS (N)	Number of significant digits for N, e.g. 31.
R	EPSILON (X)	Number $\ll 1$, for numbers like X, e.g. 2⋆⋆–23.
I	EXPONENT (X)	Exponent part of the model for real X.
R	FRACTION (X)	Fractional part of the model for real X.
N	HUGE (N)	Largest number for numbers like N, e.g. 2⋆⋆128.
R	NEAREST (X,Y)	Nearest number at X in the direction of sign Y.
I	RADIX (N)	Base of the model for numbers like N, e.g. 2.
I	RANGE (A)	Decimal exponent range for A, e.g. 37.
R	RRSPACING (X)	Reciprocal of relative spacing of numbers near X.
R	SCALE (X,I)	Return X times b⋆⋆I, where base b = RADIX (X).
R	SET_EXPONENT (X,I)	Real with mantissa part of X and exponent part of I.
R	SPACING (X)	Absolute spacing of numbers near X, e.g. 2⋆⋆-17.
R	TINY (N)	Smallest positive number, like N, e.g. 2⋆⋆–126.
NUMERIC FUNCTIONS		
A	ABS (A)	Absolute value of A.
R	AIMAG (Z)	Imaginary part of complex number.
R	ANINT (X [,K])	Real whole number nearest to X.
I	CEILING (X)	Least integer greater than or equal to real X.
Z	CMPLX (X [,Y][,K])	Convert real(s) to complex type, of given kind.
Z	CONJG (Z)	Conjugate of complex number Z.
R	DBLE (A)	Convert A to double precision real.
R	DPROD (X,Y)	Double precision real product of two real scalars.
I	FLOOR (X)	Greatest integer less than or equal to X.
I	INT (A [,K])	Convert A to integer type, of given kind.
N	MAX (N,N_2 [,N_3,...])	Maximum value of two or more numbers same type.
N	MIN (N,N_2 [,N_3,...])	Minimum value of two or more same type numbers.
N	MOD (N,N_2)	Remainder for N_2, *i.e.*, N-INT(N/N_2)⋆N_2.
N	MODULO (N,N_2)	Modulo, N-FLOOR(N/N_2)⋆N_2.
R	REAL (A [,K])	Convert A to real type, of type K.
N	SIGN (N,N_2)	Absolute value of N times sign of same type N_2.
PADDING		
B	ISHFT (I,I_SHIFT)	Logical shift of bits of I by I_SHIFT, pad with 0.

(continued)

Subject Table of Fortran 90 Intrinsic Functions (continued)		
Type	**Intrinsic**	**Description**
⋆,V	PACK (⋆_ARRAY,M [,V_PAD])	Pack ARRAY for true M into vector, pad from V_PAD.
⋆	RESHAPE (⋆_ARRAY,I_V_SHAPE [,⋆_PAD] [,V_ORDER])	Reshape ARRAY to vector SHAPE, pad, re-order.
POINTER		
L	ASSOCIATED (P [,T])	True if pointer, P, is associated with any target, or T.
PRESENCE		
L	PRESENT (OPTIONAL)	True if optional argument is present in call.
RANDOM NUMBER		
call	RANDOM_NUMBER (X)	Pseudo-random numbers in range $0 < X < 1$.
call	RANDOM_SEED ([I_SIZE] [,I_V_P][,I_V_G])	Initialize random number generator.
REALS		
R	AINT (X [,K])	Truncate X to a real whole number.
R	ANINT (X [,K])	Real whole number nearest to X.
R	AMAX0 (I,I2 [,I3])	Maximum real from list of integers.
R	AMIN0 (I,I2 [,I3])	Minimum real from list of integers.
R	REAL (A [,K])	Convert A to real type, of type K.
I	SELECTED_REAL_KIND ([I] [,I_r])	Real with precision, I, and exponent range, I_r.
REDUCTION		
L	ALL (M [,D])	True if all mask, M, terms are true, along D.
L	ANY (M [,D])	True if any mask, M, term is true, along D.
I	COUNT (M [,D])	Number of true mask, M, terms, along dimension D.
N	MAXVAL (N_ARRAY [,D] [,M])	Value of max ARRAY term, along D, passing M.
N	MINVAL (N_ARRAY [,D] [,M])	Value of min ARRAY term, along D, passing M.
A	PRODUCT (A_ARRAY [,D] [,M])	Product of ARRAY terms, along D, for mask M.
A	SUM (A_ARRAY [,D] [,M])	Sum of ARRAY terms, along D, passing mask M.
RESHAPING ARRAYS		
⋆	CSHIFT (⋆_ARRAY,I_SHIFT [,D])	Perform circular shift out and in for I_SHIFT terms.
⋆	EOSHIFT (⋆_ARRAY,I_SHFT [,⋆_FIL] [,D])	End-off shift ARRAY, and fill, in dimension D.
⋆,V	PACK (⋆_ARRAY,M [,V_PAD])	Pack ARRAY for true M into vector, pad from V_PAD.
⋆	RESHAPE (⋆_ARRAY,I_V_SHAPE [,⋆_PAD] [,V_ORDER])	Reshape ARRAY to vector SHAPE, pad, re-order.
⋆	UNPACK (V,M,⋆_USE)	Unpack V for true elements of M, into USE.
REVERSE ORDER		
I	INDEX (S,S_SUB [,L_BACK])	Left starting position of S_SUB within S (rightmost).
I	SCAN (S,S_SET [,L_BACK])	Left character index in S found in S_SET; (rightmost).
I	VERIFY (S,S_SET [,L_BACK])	First position in S not found in S_SET, (or last).
SHIFTS		
⋆	CSHIFT (⋆_ARRAY,I_SHIFT [,D])	Perform circular shift out and in for I_SHIFT terms.
⋆	EOSHIFT (⋆_ARRAY,I_SHIFT [,⋆_FILL][,D])	Perform end-off shift, and fill, in dimension D.
B	ISHFT (I,I_SHIFT)	Logical shift of bits of I by I_SHIFT, pad with 0.
B	ISHFTC (I,I_SHIFT [,I_SIZE])	Logical circular shift of I_SIZE rightmost bits of I.

(continued)

Subject Table of Fortran 90 Intrinsic Functions (continued)		
Type	**Intrinsic**	**Description**
STRING		
C	ADJUSTL (S)	Adjust S left, move leading blanks to trailing blanks.
C	ADJUSTR (S)	Adjust S right, move trailing to leading blanks.
I	INDEX (S,S_SUB [,L_BACK])	Left starting position of S_SUB within S (or right).
I	LEN (S)	Total character string length.
I	LEN_TRIM (S)	Length of S without trailing blanks.
L	LGE (S,S_2)	True if S is $\geq$ to S_2 in ASCII collating sequence.
L	LGT (S,S_2)	True if S follows S_2 in ASCII collating sequence.
L	LLE (S,S_2)	True if S is $\leq$ to S_2 in ASCII collating sequence.
L	LLT (S,S_2)	True if S precedes S_2 in ASCII collating sequence.
S	REPEAT (S,I_COPIES)	Concatenates I_COPIES of string S.
I	SCAN (S,S_SET [,L_BACK])	Left character index in S found in S_SET; (or right).
S	TRIM (S)	Remove trailing blanks from a single string.
I	VERIFY (S,S_SET [,L_BACK])	First position in S not found in S_SET, (or last).
TARGET		
L	ASSOCIATED (P [,T])	True if pointer, P, is associated with any target, or T.
TIME		
call	DATE_AND_TIME ([S_DATE] [,S_TIME] [,S_ZONE] [,I_V_VALUES])	Real-time clock data.
call	SYSTEM_CLOCK ([I_NOW] [,I_RAT] [,I_MX])	Integer data from real-time clock.
VECTOR (See ARRAYS)		
N,L	DOT_PRODUCT (V,V_2)	Dot product of vectors V and V_2.
I,V	LBOUND ($\star$_ARRAY [,D])	ARRAY lower bound(s) vector, along D.
I,V	MAXLOC (N_ARRAY [,M])	Location(s) of maximum ARRAY term, passing M.
I,V	MINLOC (N_ARRAY [,M])	Location(s) of minimum ARRAY term, passing M.
$\star$,V	PACK ($\star$_ARRAY,M [,V_PAD])	Pack ARRAY for true M into vector, pad from V_PAD.
$\star$	RESHAPE ($\star$_ARRAY,I_V_SHAPE [,$\star$_PAD] [,V_ORDER])	Reshape ARRAY to vector SHAPE, pad, re-order.
I,V	SHAPE ($\star$_ARRAY)	ARRAY (or scalar) shape vector.
$\star$	TRANSFER ($\star$_ARRAY, V_MOLD [,I_SIZE])	Same representation as ARRAY, but type of MOLD.
I,V	UBOUND ($\star$_ARRAY [,D])	ARRAY upper bound(s) vector, along dimension D.

B.3 Syntax of Fortran 90 Statements

The following is a list of the recommended Fortran90 statements. Additional statements are allowed, but have been declared obsolete, and are expected to be deleted in future standards. Thus, they should not be utilized in new programs. They are appended to the end of this list. Below we list the standard syntax for the Fortran90 statements. In some cases the most common simple form of a statement is shown before it's more general options. Such optional features are shown included in brackets, `[ ]`, and a vertical bar | means "or." Note that the new attribute terminator symbol `::` is always optional, but its use is recommended.

The following abbreviations are employed: arg=argument, attr=attribute, exp=expression, i_=integer, r_=real, s_=string, spec=specifier, and here [type] means CHARACTER | COMPLEX | INTEGER | LOGICAL | REAL, or a user defined name given in a TYPE statement. Recall that F90 allows variable names to be 31 characters long and they may include an underscore (but F77 allows only 6 characters and no underscore). F90 lines may contain up to 132 characters (but just 72 in F77). All

	MATLAB	C++	F90
Pre-allocate linear array	`A(100)=0`	`int A[100];`[a]	`integer A(100)`
Initialize to a constant value of 12	`for j=1:100 % slow` `  A(j)=12` `  end` `% better way` `A=12*ones(1,100)`	`for (j=0; j<100; j++)` `  A[j]=12;`	`A=12`
Pre-allocate two-dimensional array	`A=ones(10,10)`	`int A[10][10];`	`integer A(10,10)`

[a] C++ has a starting subscript of 0, but the argument in the allocation statement is the array's size.

Table B.41: Array initialization constructs.

Action	MATLAB	C++	F90
Define size	`A=zeros(2,3)`[a]	`int A[2][3];`	`integer,dimension(2,3)::A`
Enter rows	`A=[1,7,-2;` `  3, 4, 6];`	`int A[2][3]={` `  {1,7,2},` `  {3,4,6}` `};`	`A(1,:)=(/1,7,-2/)` `A(2,:)=(/3,4,6/)`

[a] Optional in MATLAB, but improves efficiency.

Table B.42: Array initialization constructs.

standard F77 statements are a sub-set of F90. Attribute options, and their specifiers, for each statement are given in the companion table "Fortran 90 Attributes and Specifiers". The numerous options for the INQUIRE statement are given in the table entitled "Options for F90 INQUIRE."

In addition to the statements given below F90 offers intrinsic array operations, implied do loops, vector subscripts, and about 160 intrinsic functions. Those functions, with their arguments, are given in tables "Alphabetical Table of Fortran 90 Intrinsic Functions and Subroutines," and "Subject Table of Fortran 90 Intrinsic Functions and Subroutines."

F90 Syntax

! preceeds a comment in F90
in column one denotes a comment line in F77
& continues a line in F90 (must be in column 6 for F77)
; terminates a statement in F90 (allows multiple statements per line)
variable = expression_or_statement ! is an assignment (column 7 in F77)

ALLOCATABLE [::] array_name[(extents)] [, array_name[(extents)]]
ALLOCATE (array_name)
ALLOCATE (array_name [, STAT=status] [,array_name [, STAT=status]])
BACKSPACE i_exp ! file unit number
BACKSPACE ([UNIT=]i_value [, IOSTAT=i_variable] [, ERR=i_label])
C in column one denotes a comment line in F77
CALL subroutine_name [([args])]
CASE (range_list) [select_name] ! purpose
CASE DEFAULT [select_name] ! purpose
CHARACTER LEN=i_value [::] s_list
CHARACTER [(LEN=i_value | * [, KIND=]i_kind)] [[, attr_list] ::] s_list
CHARACTER [(i_value | *, [KIND=]i_kind)] [[, attr_list] ::] s_list

(continued)

F90 *Syntax (continued)*

```
CHARACTER [([KIND=i_kind] [, LEN=i_value | *])] [[, attr_list] ::] s_list
CLOSE (i_value)  ! unit number
CLOSE ([UNIT=]i_value [, ERR=i_label] [, IOSTAT=i_variable] [, STATUS=exp])
COMPLEX [::] variable_list
COMPLEX [([KIND=]i_kind)] [[, attr_list] ::] variable_list
CONTAINS  ! internal definitions follow
CYCLE  ! current do only for a purpose
CYCLE [nested_do_name]  ! and terminate its sub_do's for a purpose
DEALLOCATE (array_name)
DEALLOCATE (array_name [, STAT=status] [, array_name [, STAT=status]])
DIMENSION array_name(extents) [, array_name(extents)]
DO  ! forever
DO i_variable = i_start, i_stop  ! loop_name_or_purpose
DO [i_variable = i_start, i_stop [, i_inc]]  ! loop_name_or_purpose
DO [i_label,] [i_variable = i_start, i_stop [, i_inc]]  ! loop_name
[loop_name:] DO [i_variable = i_start, i_stop [, i_inc]]  ! purpose
[loop_name:] DO [i_label,] [i_variable = i_start, i_stop [, i_inc]]
DO WHILE (logical_expression)  ! obsolete, use DO-EXIT pair
DO [i_label,] WHILE (logical_expression)  ! obsolete-obsolete
[name:] DO [i_label,] WHILE (logical_expression)  ! obsolete
ELSE [if_name]
ELSE IF (logical_expression) THEN [if_name]
ELSE WHERE (logical_expression)
END [name]  ! purpose
END DO [do_name]  ! purpose
END FUNCTION [function_name]  ! purpose
END IF [if_name]  ! purpose
END INTERFACE  ! purpose
END MODULE [module_name]  ! purpose
END PROGRAM [program_name]  ! purpose
END SELECT [select_name]  ! purpose
END SUBROUTINE [name]  ! purpose
END TYPE [type_name]  ! purpose
END WHERE  ! purpose
ENDFILE i_exp  ! for file unit number
ENDFILE ([UNIT=]i_value [, IOSTAT=i_variable] [, ERR=i_label])
ENTRY entry_name [([args])] [RESULT(variable_name)]
EXIT  ! current do only for a purpose
EXIT [nested_do_name]  ! and its sub_do's for a purpose
EXTERNAL program_list
i_label FORMAT (specification_and_edit_list)
FUNCTION name ([args])  ! purpose
FUNCTION name ([args]) [RESULT(variable_name)]  ! purpose
[type] [RECURSIVE] FUNCTION name ([args]) [RESULT(variable_name)]
[RECURSIVE] [type] FUNCTION name ([args]) [RESULT(variable_name)]
GO TO i_label  ! for_a_reason
IF (logical_expression) executable_statement
[name:] IF (logical_expression) THEN  ! state_purpose
IMPLICIT type (letter_list)  ! F77 (a-h,o-z) real, (i-n) integer
IMPLICIT NONE  ! F90 recommended default
INCLUDE source_file_path_name  ! purpose
INQUIRE ([FILE=]'name_string' [, see_INQUIRE_table])  ! re file
```

(continued)

F90 *Syntax (continued)*

```
INQUIRE ([NAME=]s_variable [, see_INQUIRE_table])  ! re file
INQUIRE (IOLENGTH=i_variable [, see_INQUIRE_table])  ! re output
INQUIRE ([UNIT=]i_value [, see_INQUIRE_table])  ! re unit
INTEGER [::] variable_list
INTEGER [([KIND=]i_kind)] [[, attr_list] ::] variable_list
INTENT ([IN | INOUT | OUT]) argument_list
INTERFACE ASSIGNMENT (+ | - | * | / | = | **)  ! user extension
INTERFACE OPERATOR (.operator.)  ! user defined
INTERFACE [interface_name]
INTRINSIC function_list
LOGICAL [::] variable_list
LOGICAL [([KIND=]i_kind)] [[, attr_list] ::] variable_list
MODULE PROCEDURE program_list
MODULE module_name  ! purpose
NULLIFY (pointer_list)
OPEN (i_value)  ! unit number
OPEN ([UNIT=]i_value [, ERR=i_label] [, IOSTAT=i_variable] [, other_spec])
OPTIONAL [::] argument_list
PARAMETER (variable=value [, variable=value])
POINTER [::] name[(extent)] [, name[(extent)]]  ! purpose
PRINT * , output_list  ! default free format
PRINT * , (io_implied_do)  ! default free format
PRINT '(formats)' , output_list  ! formatted
PRINT '(formats)' , (io_implied_do)  ! formatted
PRIVATE [[::] module_variable_list]  ! limit access
PROGRAM [program_name]  ! purpose
PUBLIC [[::] module_variable_list]  ! default access
READ * , input_list  ! default free format
READ * , (io_implied_do)  ! default free format
READ '(formats)', input_list  ! formatted
READ '(formats)', (io_implied_do)  ! formatted
READ ([UNIT=]i_value, [FMT=]i_label [, io_spec_list]), input_list  ! formatted
READ ([UNIT=]i_value, s_variable [, io_spec_list]), input_list  ! formatted
READ ([UNIT=]i_value, '(formats)' [, io_spec_list]), input_list  ! formatted
READ (i_value), input_list  ! binary read
READ ([UNIT=]i_value, [, io_spec_list]), input_list  ! binary read
READ (s_variable, [FMT=]i_label), input_list  ! internal file type change
READ ([UNIT=]s_variable, [FMT=]i_label [, io_spec_list]), input_list  ! internal file change
REAL [::] variable_list
REAL [([KIND=]i_kind)] [[, attr_list] ::] variable_list
RECURSIVE FUNCTION name ([args]) [RESULT(variable_name)]  ! purpose
[type] RECURSIVE FUNCTION name ([args]) [RESULT(variable_name)]  ! purpose
RECURSIVE SUBROUTINE name [([args])]  ! purpose
RETURN  ! from subroutine_name
REWIND i_exp  ! file unit number
REWIND ([UNIT=]i_value [, IOSTAT=i_variable] [, ERR=i_label])
SAVE [[::] variable_list]
[name:] SELECT CASE (value)
SEQUENCE
STOP ['stop_message_string']
SUBROUTINE name [([args])]  ! purpose
SUBROUTINE name [([args])] [args, optional_args]  ! purpose
```

(continued)

F90 *Syntax (continued)*

```
[RECURSIVE] SUBROUTINE name [([args])]  ! purpose
TARGET [::] name[(extent)] [, name[(extent)]]
TYPE (type_name) [[, attr_list] ::] variable_list
TYPE [, PRIVATE | PUBLIC] name
USE module_name [, ONLY: list_in_module_name]  ! purpose
USE module_name [, new_var_or_sub=>old_name]  ! purpose
WHERE (logical_array_expression)  ! then
WHERE (logical_array_expression) array_variable = array_expression
WRITE * , output_list  ! default free format
WRITE * , (io_implied_do)  ! default free format
WRITE '(formats)', output_list  ! formatted write
WRITE '(formats)', (io_implied_do)  ! formatted write
WRITE ([UNIT=]i_value, [FMT=]i_label [, io_spec_list]), output_list  ! formatted write
WRITE ([UNIT=]i_value, s_variable [, io_spec_list]), output_list  ! formatted write
WRITE ([UNIT=]i_value, '(formats)' [, io_spec_list]), output_list  ! formatted write
WRITE (i_value), output_list  ! binary write
WRITE (i_value), (io_implied_do)  ! binary write
WRITE ([UNIT=]i_value, [, io_spec_list]), output_list  ! binary write
WRITE (s_variable, [FMT=]i_label), output_list  ! internal file type change
WRITE ([UNIT=]s_variable, [FMT=]i_label [, io_spec_list]), output_list  ! internal file change
```

Obsolescent statements are those from Fortran77 that are redundant and for which better methods are available in both Fortran77 and Fortran90.

Obsolete Syntax

```
ASSIGN i_label TO i_variable
BLOCK DATA [block_data_name]
COMMON [/common_block_name/] r_variable_list, i_variable_list
[i_label] CONTINUE  ! from do [do_name]
DATA variable_list / value_list /
DATA (array_implied_do) / value_list /
DOUBLE PRECISION [[, attr_list] :: ] variable_list
DO [i_label,] [r_variable = r_start, r_stop [, r_inc]]  ! real control
DO_CONTINUE_pair
[name:] DO [i_label,] WHILE (logical_expression)  ! obsolete
END BLOCK DATA [block_data_name]
EQUIVALENCE (variable_1, variable_2) [ , (variable_3, variable_4)]
GO TO (i_label_1,i_label_2,...,i_label_n)[ , ] i_variable
IF (arithmetic_exp) i_label_neg, i_label_zero, i_label_pos
NAMELIST /group_name/ variable_list
PAUSE  ! for human action
RETURN alternates
statement_function (args) = expression
```

The attributes lists for the type declarations, e.g. `REAL`, are `ALLOCATABLE`, `DIMENSION`, `INTENT`, `OPTIONAL`, `KIND`, `POINTER`, `PARAMETER`, `PRIVATE`, `PUBLIC`, `SAVE`, and `TARGET`; those for `OPEN` and `CLOSE` are `ACCESS`, `ACTION`, `BLANK`, and `DELIM`; while those for `READ` and `WRITE` are `ADVANCE`, `END`, `EOR`, `ERR`, and `FMT`.

	MATLAB	C++	F90
Addition $\mathbf{C} = \mathbf{A} + \mathbf{B}$	`C=A+B`	`for (i=0; i<10; i++){` `  for (j=0; j<10; j++){` `  C[i][j]=A[i][j]+B[i][j];` `  }` `  }`	`C=A+B`
Multiplication $\mathbf{C} = \mathbf{AB}$	`C=A*B`	`for (i=0; i<10; i++){` `  for (j=0; j<10; j++){` `  C[i][j] = 0;` `  for (k=0; k<10; k++){` `  C[i][j] += A[i][k]*B[k][j];` `  }` `  }` `  }`	`C=matmul(A,B)`
Scalar multiplication $\mathbf{C} = a\mathbf{B}$	`C=a*B`	`for (i=0; i<10; i++){` `  for (j=0; j < 10; j++){` `  C[i][j] = a*B[i][j];` `  }` `  }`	`C=a*B`
Matrix inverse $\mathbf{B} = \mathbf{A}^{-1}$	`B=inv(A)`	[a]	`B=inv(A)`[a]

[a]Neither C++ nor F90 have matrix inverse functions as part of their language definitions nor as part of standard collections of mathematical functions (like those listed in Table 4.7). Instead, a special function, usually drawn from a library of numerical functions, or a user defined operation, must be used.

Table B.43: Elementary matrix computational routines.

C++	`int*  point, vector, matrix` `...` `point = new type_tag` `vector = new type_tag [space_1]` `if (vector == 0) {error_process}` `matrix = new type_tag [space_1 * space_2]` `...` `delete matrix` `...` `delete vector` `delete point`
F90	`type_tag, pointer, allocatable :: point` `type_tag, allocatable :: vector (:), matrix (:,:)` `...` `allocate (point)` `allocate (vector (space_1), STAT = my_int)` `if (my_int /= 0) error_process` `allocate (matrix (space_1, space_2))` `...` `deallocate (matrix)` `if (associated (point, target_name)) pointer_action...` `if (allocated (matrix)) matrix_action...` `...` `deallocate (vector)` `deallocate (point)`

Table B.44: Dynamic allocation of arrays and pointers.

```
SUBROUTINE AUTO_ARRAYS (M,N, OTHER)
USE GLOBAL_CONSTANTS ! FOR INTEGER K
   IMPLICIT NONE
   INTEGER, INTENT (IN) :: M,N
   type_tag, INTENT (OUT) :: OTHER (M,N) ! dummy array

!  Automatic array allocations
   type_tag :: FROM_USE (K)
   type_tag :: FROM_ARG (M)
   type_tag :: FROM_MIX (K,N)
   ...
!  Automatic deallocation at end of scope
END SUBROUTINE AUTO_ARRAYS
```

Table B.45: Automatic memory management of local scope arrays.

```
module derived_class_name
      use base_class_name
! new attribute declarations, if any
                ...
contains


    ! new member definitions
                ...
end module derived_class_name
```

Table B.46: F90 Single Inheritance Form.

```
module derived_class_name
      use base_class_name, only: list_of_entities
! new attribute declarations, if any
                ...
contains


    ! new member definitions
                ...
end module derived_class_name
```

Table B.47: F90 Selective Single Inheritance Form.

```
module derived_class_name
      use base_class_name, local_name => base_entity_name
! new attribute declarations, if any
                ...
contains


    ! new member definitions
                ...
end module derived_class_name
```

Table B.48: F90 Single Inheritance Form, with Local Renaming.

```
module derived_class_name
      use  base1_class_name
      use  base2_class_name
      use  base3_class_name, only: list_of_entities
      use  base4_class_name, local_name => base_entity_name
! new attribute declarations, if any
             . . .
contains


      ! new member definitions
             . . .
end module derived_class_name
```

Table B.49: F90 Multiple Selective Inheritance with Renaming.

Examples of F90 Statements

The following is a list of examples of the recommended Fortran90 statements. Some have been declared obsolete, and are expected to be deleted in future standards. Thus, they should not be utilized in new programs. They are noted in the comments. In some cases the most common simple form of a statement is shown along with it's more general options. Note that the new attribute terminator symbol :: is always optional, but its use is recommended. While Fortran is not case-sensitive, this table employs upper case letters to denote standard features, and lower case letters for user supplied information. The following abbreviations are employed: arg=argument, attr=attribute, exp=expression, i_=integer, l_=logical, r_=real, s_=string, spec=specifier, z_=complex.

Recall that F90 allows variable names to be 31 characters long and they may include an underscore (but F77 allows only six characters and no underscore). F90 lines may contain up to 132 characters (but just 72 in F77). All standard F77 statements are a sub-set of F90.

The attributes lists for the type declariations, e.g. `REAL`, are `ALLOCATABLE, DIMENSION, INTENT, OPTIONAL, KIND, POINTER, PARAMETER, PRIVATE, PUBLIC, SAVE`, and `TARGET`. Those optional attributes for `OPEN` are `ACCESS` = [`DIRECT, SEQUENTIAL`], `ACTION` = [`READ, READWRITE, WRITE`], `BLANK` = [`NULL, ZERO`], `DELIM` = [`APOSTROPHE, NONE, QUOTE`], `ERR` = i_label, `FILE` = s_name, `FORM` = [`FORMATTED, UNFORMATTED`], `IOSTAT` = i_var, `PAD` = [`NO, YES`], `POSITION` = [`APPEND, ASIS, REWIND`], `RECL` = i_len, `STATUS` = [`NEW, OLD, REPLACE, SEARCH, UNKNOWN`], and `UNIT` = i_unit; while `CLOSE` utilizes only `ERR, IOSTAT, STATUS`, and `UNIT`.

The io_spec_list options for `READ` and `WRITE` are `ADVANCE` = [`NO, YES`], `END` = i_label, `EOR` = i_label, `ERR` = i_label, `FMT` = [*, i_label, s_var], `IOSTAT` = i_var, `NML` = var_list, `REC` = i_exp, `SIZE` = i_size, and `UNIT` = i_unit.

Fortran Statement Examples

Name	*Examples*	*Comments*
Allocatable	ALLOCATABLE :: force, stiffness	By name
	ALLOCATABLE :: force(:), stiffness(:,:)	Ranks
Allocate	ALLOCATE (hyper_matrix(5, 10, 3))	
	ALLOCATE (force(m))	
	ALLOCATE (array_name(3, 3, 3, 3), STAT=i_err)	Error status
Assign	ASSIGN 9 TO k	Obsolete
Assignment	c = 'b'	Character
	s = "abc"	String
	s = c // 'abc'	Concatenation
	s = string(j:m)	Sub-string
	s_fmt = '(2F5.1)'	Stored format
	l = l_1 .OR. l_2	Logical
	l = m < = 80	
	poor = (final > = 60) .AND. (final < 70)	
	proceed = .TRUE.	
	n = n + 1	Arithmetic
	x = b'1010'	Binary
	z = (0.0, 1.0)	Complex
	r = SQRT (5.)	Function
	converged = (ABS (x0 – x) < 2*SPACING (x))	
	x = z'B'	Hexadecimal
	k = 123	Integer
	x = o'12'	Octal
	r = 321.	Real
	a = 23. ; j = 120 ; ans = .TRUE.;	Semicolon
	k = SELECTED_INTEGER_KIND (20)	Kind
	m = SELECTED_REAL_KIND (16, 30)	

(continued)

Fortran Statement Examples (continued)		
Name	*Examples*	*Comments*
	long = SELECTED_REAL_KIND (9, 20) pi = 3.1459265_long	
	a = b + c d = MATMUL (a, b) e = TRANSPOSE (d) f = 0 ; g = (/ 2. , 4. , 6. /) B = Al:, n:(:–1) x = (/ (k, k = 0, n) /) * d	Matrix add Matrix multiply Matrix transpose Matrix initialize Matrix flipped Implied do
	kth_row => a(k,:) corners => a(1:n:(n-1), 1:m:(m-1)) p_2 => r	Pointer
	student_record%rank = 51 patient_data%city = 'houston'	Defined type
	sqrt(x) = DSQRT(x) ! function statement	Obsolete
Backspace	BACKSPACE i_exp BACKSPACE 8 BACKSPACE (UNIT=9, IOSTAT=i, ERR=5) BACKSPACE (9, IOSTAT=io_ok, ERR=99) BACKSPACE (UNIT=9, IOSTAT=io_ok, ERR=99) BACKSPACE (8, IOSTAT=io_ok)	Compute unit Unit Error go to I/O status
Block Data	BLOCK DATA ! Obsolete BLOCK DATA winter ! Obsolete	 Named
C	C in column one denotes a comment line in F77 * in column one denotes a comment line in F77 ! anywhere starts a comment line in F90	Obsolete Obsolete
Call	CALL sub1 (a, b) CALL sub2 (a, b, *5) ! Obsolete, use CASE CALL sub3 CALL subroutine_name (args, optional_args)	 Alt return to 5 No arguments Optional arg
Case	CASE (range_list) CASE (range_list) select_name	See SELECT Named
Case Default	CASE DEFAULT CASE DEFAULT select_name	See SELECT Named
Character	CHARACTER (80) s, s_2*3(4) CHARACTER *16 a, b, c CHARACTER * home_team CHARACTER (*), INTENT(IN) :: home_team CHARACTER (LEN=3) :: b = 'xyz' CHARACTER LEN=40 :: monday, wednesday, friday CHARACTER (LEN=40), attr_list :: last, first, middle CHARACTER (40), attr_list :: name, state CHARACTER (*), PARAMETER :: reply = "Invalid Data" CHARACTER (*, KIND=greek), attr_list :: s1_list CHARACTER (*, KIND=greek), attr_list :: last, first, middle CHARACTER (KIND=cyrillic, LEN=40) :: name, state CHARACTER (KIND=cyrillic, *), attr_list :: s_list	 :: recommended Intent Initialize b Kind
Close	CLOSE (7) CLOSE (UNIT=k) CLOSE (UNIT=8, ERR=90, IOSTAT=i) CLOSE (8, ERR=99, IOSTAT=io_ok, STATUS='KEEP')	Unit number Error go to I/O status

(continued)

Fortran Statement Examples (continued)		
Name	*Examples*	*Comments*
	CLOSE (9, ERR=99, IOSTAT=io, STATUS='DELETE') CLOSE (UNIT=8, ERR=95, IOSTAT=io_ok)	File status
Common	COMMON / name / h, p, t ! Obsolete COMMON p, d, q(m,n) ! Obsolete	Named common Blank common
Complex	COMPLEX u, v, w(3, 6) COMPLEX :: u = (1.0,1.0), v = (1.0,10.0) COMPLEX :: variable_list COMPLEX attr_list :: variable_list COMPLEX (KIND=i2_kind), attr_list :: variable_list	:: recommended Initialize u and v Kind
Contains	CONTAINS	Internal definitions
	CONTAINS FUNCTION mine (b) . . . END FUNCTION mine	Or subroutines
Continuation	! any non-block character in column 6 flags continuation & at the end flags continuation to next line & at the beginning flags continuation from above line	F77 obsolete F90 standard
	a_long_name = a_constant_value* & another_value ! on following line	
	a_long_name_here_is_set_to = value & * another_value ! continued from above	
Continue	100 CONTINUE	Obsolete
Cycle	CYCLE CYCLE nested_do_name	Current do only Terminate sub_dos
Data	DATA a, s / 4.01, 'z' / DATA s_fmt / '(2F5.1)' / DATA (r(k), k=1,3) / 0.7, 0.8, 1.9 / DATA array (4,4) / 1.0 / DATA bit_val / b'0011111' /	Obsolete Stored format Implied do Single value Binary
Deallocate	DEALLOCATE (force) DEALLOCATE (force, STAT=i_err)	File name Error status
Dimension	DIMENSION array (4, 4) DIMENSION v(1000), w(3) = (/ 1., 2., 4. /) DIMENSION force(20), stiffness(:,:) DIMENSION (5,10,3) :: triplet	 Initialize w :: recommended
	INTEGER, DIMENSION (:,:) :: material, nodes_list REAL, DIMENSION(m, n) :: a, b REAL, DIMENSION (:,:) :: force, stiffness `REAL, DIMENSION` (5,10,3), `INTENT(IN)` :: triplet	Typed Intent
Do	DO 100 j = init, last, incr ! Obsolete . . . 100 CONTINUE	Labeled do Obsolete
	DO j = init, last . . . END DO	Unlabeled do
	DO ! forever . . . END DO ! forever	Unlabeled do
	DO WHILE (diff <= delta) . . . END DO	Unlabeled while
	DO 100 WHILE (diff < = delta) ! Obsolete	Labeled while

(continued)

Fortran Statement Examples (continued)		
Name	*Examples*	*Comments*
	. . . 100 CONTINUE	 Obsolete
	DO	Forever
	DO k = i_start, i_stop DO k = i_start, i_stop, i_inc DO 10, k = i_start, i_stop	Integer range Increment Obsolete
	do_name: DO k = i_start, i_stop, i_inc do_name: DO 10, k = i_start, i_stop, i_inc	Named Named label
	DO 10, r_variable = r_start, r_stop, r_inc ! Obsolete	Real range
Do While	DO WHILE (.NOT. converged) DO 10, WHILE (.NOT. converged) do_name: DO 10, WHILE (.NOT. converged)	Use DO-EXIT pair Obsolete Obsolete
Double Precision	DOUBLE PRECISION a, d, y(2) DOUBLE PRECISION :: a, d = 1.2D3, y(2) DOUBLE PRECISION, attr_list :: variable_list	Obsolete Initialize D Obsolete
Else	ELSE ELSE leap_year	Then Named
Else If	ELSE IF (k > 50) THEN ELSE IF (days_in_year == 364) THEN ELSE IF (days_in_year == 364) THEN leap_year	 Named
Elsewhere	ELSEWHERE	See WHERE
End	END END name	 Named
End Block Data	END BLOCK DATA END BLOCK DATA block_data_name	Obsolete Obsolete
End Do	END DO END DO do_name	 Named
End Function	END FUNCTION function_name END FUNCTION	
End If	END IF leap_year END IF	Named
End Interface	END INTERFACE	
End Module	END MODULE my_matrix_operators END MODULE	
End Program	END PROGRAM program_name END PROGRAM	
End Select	END SELECT select_name END SELECT	Named
End Subroutine	END SUBROUTINE name END SUBROUTINE	
End Type	END TYPE type_name END TYPE	See TYPE
End Where	END WHERE	See WHERE
Endfile	ENDFILE i_exp ENDFILE (UNIT=k) ENDFILE k ENDFILE (UNIT=8, ERR=95) ENDFILE (7, IOSTAT=io_ok, ERR=99) ENDFILE (UNIT=8, IOSTAT=k, ERR=9) ENDFILE (UNIT=9, IOSTAT=io_ok, ERR=99)	Compute unit Unit number Error go to I/O status

(continued)

Fortran Statement Examples (continued)		
Name	*Examples*	*Comments*
Entry	ENTRY sec1 (x, y) ENTRY sec2 (a1, a2, *4) ! Obsolete, use CASE ENTRY section ENTRY entry_name RESULT(variable_name)	Arguments Alternate return to 4 No arguments Result
Equivalence	EQUIVALENCE (v (1), a (1,1)) EQUIVALENCE (v, a) EQUIVALENCE (x, v(10)), (p, q, d)	Obsolete
Exit	EXIT EXIT nested_do_name	Current do only Current & sub-dos
External	EXTERNAL my_program	
Format	10 FORMAT (2X, 2I3, 3F6.1, 4E12.2, 2A6, 3L2) 10 FORMAT (// 2D6.1, 3G12.2) 10 FORMAT (2I3.3, 3G6.1E3, 4E12.2E3) 10 FORMAT ('a quoted string', "another", I2) 10 FORMAT (1X, T10, A1, T20, A1) 10 FORMAT (5X, TR10, A1, TR10, A1, TL5, A1) 10 FORMAT ("Init=", I2, :, 3X, "Last=", I2) 10 FORMAT ('Octal ', o6, ', Hex ' z6) 10 FORMAT (specification_and_edit_list)	X I F E A L D, G Exponent w Strings Tabs Tab right, left **:** stop if empty Octal, hex
Function	FUNCTION z (a, b) FUNCTION w (e, d) RESULT (a) FUNCTION name (args) FUNCTION name FUNCTION name (args) RESULT(variable_name)	Arguments Result No argument
	INTEGER FUNCTION n (j, k) INTEGER FUNCTION name (args) COMPLEX RECURSIVE FUNCTION dat (args) RECURSIVE REAL FUNCTION name (args)	Type
Go To	GO TO 99	Unconditional
	GO TO (10,20,35,95), i_variable ! Obsolete	Computed
If	IF (arithmetic_exp) 95, 10, 20 ! Obsolete	Arithmetic
	IF (logic) RETURN IF (logic) n = n + 2	Logical if
	IF (logic) THEN n = n + 1 k = k + 1 END IF	if block
	leap_year: IF (logical_expression) THEN	Named
	IF (logic) THEN n = n + 1 ELSE k = k + 1 END IF	if else block
	IF (c == 'a') THEN na = na + 1 CALL sub_a ELSE IF (c == 'b') THEN nb = nb + 1 ELSE IF (c == 'c') THEN nc = nc + 1	if else-if block (Use CASE)

(continued)

Fortran Statement Examples (continued)		
Name	*Examples*	*Comments*
	CALL sub_c	
	END IF	
Implicit	IMPLICIT INTEGER (i-n)	F77 default
Type	IMPLICIT REAL (a-h,o-z)	F77 default
	IMPLICIT NONE	Recommended F90
	IMPLICIT CHARACTER *10 (f,l)	Character
	IMPLICIT COMPLEX (a-c,z)	Complex
	IMPLICIT TYPE (color) (b,g,r)	Derived type
	IMPLICIT LOGICAL (KIND=bit) (m)	Logical
Include	INCLUDE 'path/source.f'	
Inquire	INQUIRE (UNIT=3, OPENED=t_or_f)	Opened
	INQUIRE (FILE='mydata', EXIST=t_or_f)	Exists
	INQUIRE (UNIT=3, OPENED=ok, IOSTAT=k)	I/O status
	INQUIRE (FILE='name_string', see_INQUIRE_table)	Re file
	INQUIRE (NAME=s_variable, see_INQUIRE_table)	Re file
	INQUIRE (IOLENGTH=i_var, see_INQUIRE_table)	Re output
	INQUIRE (7, see_INQUIRE_table)	Re unit
	INQUIRE (UNIT=8, see_INQUIRE_table)	Re unit
Integer	INTEGER c, d(4)	:: Recommended
	INTEGER (long), attr_list :: variable_list	
	INTEGER, DIMENSION (4) :: a, d, e	
	INTEGER, ALLOCATABLE, DIMENSION(:,:) :: a, b	Allocatable
	INTEGER :: a = 100, b, c = 9	Initialize a & c
	INTEGER :: i, j, k, l, m, n, month, year = 1996	
	INTEGER, attr_list :: variable_list	
	INTEGER (KIND=i2_kind), attr_list :: variable_list	Kind
Intent	INTENT (IN) :: credit_card_owners	
	INTENT (INOUT) :: amount_due	
	INTENT (OUT) income_rank	
Interface	INTERFACE ASSIGNMENT (=)	User extension
	INTERFACE OPERATOR (+)	User extension
	INTERFACE OPERATOR (–)	User extension
	INTERFACE OPERATOR (/)	User extension
	INTERFACE OPERATOR (*)	User extension
	INTERFACE OPERATOR (**)	User extension
	INTERFACE OPERATOR (.operator.)	User defined
	INTERFACE	
	INTERFACE interface_name	
Intrinsic	INTRINSIC SQRT, EXP	Functions
Logical	LOGICAL c	:: recommended
	LOGICAL, ALLOCATABLE :: mask(:), mask_2(:,:)	Allocatable
	LOGICAL (KIND = byte) :: flag, status	Kind
	LOGICAL :: b = .FALSE., c	Initialize b
Module	MODULE PROCEDURE mat_x_mat, mat_x_vec	Generics
	MODULE my_matrix_operators	
Namelist	NAMELIST /data/ s, n, d	Obsolete
Nullify	NULLIFY (pointer_list)	
Open	OPEN (7)	Unit number
	OPEN (UNIT=3, FILE="data.test")	Name
	OPEN (UNIT=2, FILE="data", STATUS = "old")	File status

(continued)

Fortran Statement Examples (continued)		
Name	*Examples*	*Comments*
	OPEN (UNIT=3, IOSTAT=k)	I/O status
	OPEN (9, ERR = 12, ACCESS ="direct")	Access type
	OPEN (8, ERR=99, IOSTAT=io_ok)	Error go to
	OPEN (UNIT=8, ERR=99, IOSTAT=io_ok)	
Optional	OPTIONAL slow, fast	Argument list
	OPTIONAL :: argument_list	
Parameter	PARAMETER (a="xyz"), (pi=3.14159)	Character
	PARAMETER (a="z", pi=3.14159)	Real
	PARAMETER (x=11, y = x/3)	Computed
	PARAMETER, REAL :: weight = 245.6	Type
Pause	PAUSE ! for human action	Obsolete
Pointer	POINTER current, last	:: recommended
	POINTER :: name(4,5)	Rank
	REAL, POINTER :: y(:), x(:,:,:)	Type
Print	PRINT *, a, j	List-directed
	PRINT *, output_list	Default unformatted
	PRINT *, (io_implied_do)	Implied do
	PRINT *, "The squre root of", n, 'is', SQRT(n)	Function
	PRINT *, (4*k-1, k=1,10,3)	
	PRINT 10, a, j	Formatted
	PRINT 10, m_array	Array
	PRINT 10, (m(i), i = j,k)	Implied do
	PRINT 10, s(j:k)	Substring
	PRINT '(A6, I3)', a, j	Character, integer
	PRINT FMT='(A6, I3)', a, j	Included format
	PRINT data_namelist ! Obsolete	Namelist
	PRINT '(formats)', output_list	Formatted
	PRINT '(formats)', (io_implied_do)	Implied do
	PRINT '(I4)', (2*k, k=1,5)	
Private	PRIVATE	
	PRIVATE :: module_variable_list	Specific items
Program	PROGRAM my_job	
	PROGRAM	
Public	PUBLIC	
	PUBLIC :: module_variable_list	Specific items
Read	READ *, a, j	List-directed
	READ 1, a , j	Formatted
	READ 10, m_array	Formatted array
	READ 10, (m(i), i=j, k)	Implied do
	READ 10, s(i:k)	Substring
	READ '(A6, I3)', a, i	Character, integer
	READ (1, 2) x, y	Formatted file
	READ (UNIT=1, FMT=2) x, y	
	READ (1, 2, ERR=8, END=9) x, y	End of file go to
	READ (UNIT=1, FMT=2, ERR=8, END=9) x, y	Error go to
	READ (*, 2) x, y	Formatted, std out
	READ (*, 10) m_array	Unformatted array
	READ (*, 10) (m(i), i=j, k)	Implied do
	READ (*, 10) s(i:k)	Substring
	READ (1, *) x, y	Unformatted file

(continued)

Fortran Statement Examples (continued)		
Name	*Examples*	*Comments*
	READ (*, *) x, y	Unformatted, std out
	READ (1, '(A6, I3)') x, y READ (1, FMT='(A6, I3)') x, y	Character, integer Included format
	READ (1, s_fmt) x, y READ (1, FMT=s_fmt) x, y	Format in a string
	READ (*, NML=data) ! Obsolete READ (1, NML=data) ! Obsolete	Namelist read Namelist from a file
	READ (1, END=8, ERR=9) x, y	Unformatted
	READ (s2, 1, ERR=9) x	Internal, formatted
	READ (s2, *, ERR=9) x	Unformatted
	READ (s2, REC=4, END=8) x	Internal, direct
	READ (1, REC=3) v	Unformatted direct
	READ (1, 2, REC=3) v	Formatted direct
	READ *, input_list READ *, (io_implied_do) READ *, (a(j,:), j=1, rows)	Default unformatted Implied do
	READ '(formats)', input_list READ '(formats)', (io_implied_do) READ '(5I5, (5I5))', (num(k), k=1, n)	Formatted read Formatted read
	READ (8, FMT=20), input_list READ (8, FMT=20, ADVANCE='NO'), input READ (9, FMT=20, io_spec_list), input_list READ (UNIT=7, 20, io_spec_list), input_list READ (UNIT=8, FMT=10, io_spec_list), input	Formatted Advance I/O Specification
	READ (7, s_fmt, io_spec_list), input_list READ (UNIT=7, s_fmt, io_spec_list), input	Stored format
	READ (9, '(formats)', io_spec_list), input_list READ (UNIT=9, '(formats)', io_spec_list), input	Inline format
	READ (8), input_list READ (UNIT=7), input_list READ (8, io_spec_list), input_list READ (UNIT=9, io_spec_list), input_list	Binary read I/O Specification
	READ (s_variable, FMT=20), input_list READ (UNIT=s_variable, 10, io_spec_list), input	Internal file, type change
Real	REAL*4 REAL :: r, m(9) REAL*16 a, b, c REAL*8, DIMENSION(n) :: a, b, c REAL :: a = 3.14, b, c = 100.0 REAL :: variable_list REAL, attr_list :: variable_list REAL, POINTER :: a(:,:)	:: recommended Quad Precision Double Precision Initialize a & c
	REAL (KIND=i2_kind), attr_list :: variable_list REAL (double), attr_list :: variable_list	Kind
Recursive	RECURSIVE FUNCTION name RECURSIVE FUNCTION a(n) RESULT(fac) INTEGER RECURSIVE FUNCTION name (args)	Function Result
	RECURSIVE SUBROUTINE name (args) RECURSIVE SUBROUTINE name	Subroutine
Return	RETURN	Standard return

(continued)

Fortran Statement Examples (continued)		
Name	*Examples*	*Comments*
Rewind	REWIND i_exp	Compute unit
	REWIND 2	Unit number
	REWIND k	
	REWIND (UNIT=8, IOSTAT=k, ERR=9)	Error go to
	REWIND (UNIT=8, ERR=95)	
	REWIND (8, IOSTAT=io_ok, ERR=99)	I/O status
Save	SAVE a, /name/, c	Scalars, common
	SAVE	Everything
	SAVE :: variable_list	
Select Case	SELECT CASE (value)	
	name: SELECT CASE (value)	Named
	u_or_l SELECT CASE (letter) CASE ("a":"z") ! lower case lower = .TRUE. CASE ("A":"Z") ! upper case lower = .FALSE. CASE DEFAULT ! not a letter PRINT *, "Symbol is not a letter", letter lower = .FALSE. END SELECT u_or_l	Block
Sequence	SEQUENCE	Forced storage
Stop	STOP	
	STOP "invalid data"	With message
Subroutine	SUBROUTINE sub1 (a, b)	
	SUBROUTINE sub1	No arguments
	SUBROUTINE name (args, optional_args)	Optional arguments
	SUBROUTINE sub3 (a, b, *9) ! Obsolete, use CASE	Return to 9
	RECURSIVE SUBROUTINE sub2 (a, b)	Recursive
Target	TARGET :: name, name_2	See Pointer
	TARGET :: name(4,5), name_2(3)	
Type Declaration	TYPE (person) car_pool(5)	User defined type
	TYPE (color), DIMENSION(256) :: hues	
	TYPE (type_name), attr_list :: variable_list	
	TYPE (person), DIMENSION (n) :: address_book	
	TYPE (type_name) :: variable_list	
	TYPE (student_record) CHARACTER (name_len) :: last, first INTEGER :: rank END TYPE student_record	Definition block
Type Statement	TYPE, PRIVATE name	Access
	TYPE, PUBLIC :: name	
Use	USE module_name	
	USE module_name, ONLY: list_in_module_name	Only
	USE module_name, var_subr_fun_name => old_name	Rename
Where	WHERE (logical_array_mask)	Then
	WHERE (a_array > 0.0) sqrt_a = SQRT(a_array) END WHERE	Where block
	WHERE (mask > 0.0) a_array = mask	Elsewhere block

(continued)

Fortran Statement Examples (continued)		
Name	*Examples*	*Comments*
	ELSEWHERE a_array = 0.0 END WHERE	
	WHERE (a_array>0) b_array = SQRT(a_array)	Statement
Write	WRITE (*, 10) s(j:k)	Substring
	WRITE (1, *) x, y WRITE (*, *) x, y	Unformatted file Unformatted
	WRITE (1, '(A6, I3)') x, y WRITE (1, FMT='(A6, I3)') x, y	Character, integer Included format
	WRITE (1, s_fmt) x, y WRITE (1, FMT=s_fmt) x, y	Stored format string
	WRITE (*, NML=data) ! Obsolete WRITE (1, NML=data) ! Obsolete	Namelist to stdout Namelist to a file
	WRITE (1, END=8, ERR=9) x, y	Unformatted
	WRITE (1, REC=3) v	Unformatted direct
	WRITE (1, 2, REC=3) v	Formatted direct
	WRITE (s2, 1, ERR=9) x	Internal, format
	WRITE (s2, *, ERR=9) x	Unformatted
	WRITE (s2, REC=4, END=8) x	Internal, direct
	WRITE *, output_list WRITE *, (io_implied_do) WRITE *, ((a(i, j), j=1, cols), i=1, rows)	Unformatted Implied do
	WRITE '(formats)', output_list WRITE '(formats)', (io_implied_do)	Formatted write Implied do
	WRITE (7, 10, ADVANCE='NO'), output_list WRITE (8, 10, io_spec_list), output_list WRITE (9, FMT=20, io_spec_list), output_list WRITE (UNIT=7, 10, io_spec_list), output_list	Advance I/O specification
	WRITE (9, s_fmt, io_spec_list), output_list WRITE (UNIT=8, s_fmt, io_spec_list), output	Stored format
	WRITE (9, '(formats)', io_spec_list), output_list WRITE (UNIT=7, '(formats)', io_spec_list), output	Inline format
	WRITE (8), output_list WRITE (7), (io_implied_do) WRITE (8, ADVANCE='NO'), output_list WRITE (9, io_spec_list), output_list WRITE (UNIT=9, io_spec_list), output_list	Binary write Implied do Advance I/O specification
	WRITE (s_variable, FMT=20), output_list WRITE (UNIT=s_variable, FMT=20), output_list WRITE (s_variable, 20, io_spec_list), output_list WRITE (UNIT=s_var, FMT=20, io_spec), output	Internal file I/O specification