Fortran Quick Reference/Cheat Sheet

Remember: FORTRAN 77 and below is case sensitive. Fortran 90 and above is NOT case sensitive.

Introduction

Important things to note are:

- Fortran can perform array arithmetic operations.
- Spaces are ignored?
- Fortran is a compiled language which is compiled into an executable
- Blue text indicates a feature which is available from Fortran 90 onwards.
- Purple text indicates a feature which is available from Fortran 95 onwards.
- Red text indicates a feature which is available from Fortran 2003 onwards.

Terminology

Statement - An instruction which is either executabe or nonexecutable.
Construct - A sequence of statements ending with a construct terminal statement.
Function - A procedure that returns the value of a single variable.
Procedure - Either a function or subroutine. Intrinsic procedure, external procedure, module procedure, internal procedure, dummy procedure or statement function.
Subroutine - A procedure that is invoked by a CALL statement or defined assignment statement. It can return more than one argument.

Special Characters

`' (Apostrophe)`	Editing, declaring a string
`" (Quotation Marks)`	Declaring a string
`* (Asterisk)`	Comment lines.
`: (Colon)`	Editing.
`:: (Double Colon)`	Separator.
`! (Exclamation)`	inline comment.
`/ (Slash)`	Skip a line in a fmt statment?
`; (Semicolon)`	Separates Statement on single source line. Except when it is in a character context, a comment or in line 6.
`+ (Plus)`	Arithmetic operator.
`& (Ampersand)`	Line continuation charachter.(Must be in line 7 of fixed format F77. For F90 can be anywhere after the line.

Concepts and Elements

Concept	Statements
Module	Module Contains Private Public End Module Use
Interface Block	Interface Module Procedure End Interface
Derived data type	Derived type Private Sequence End Type
Subprogram	Function Subroutine Entry Contains Return
Input/Output	Backspace Close Endfile Format Inquire Open Print Read Rewind Write

Flow Control

Group	Statements

IF	IF
	ELSE IF
	ELSE
	ENDIF

CASE	SELECT CASE CASE END SELECT
Do/Do while	DO DO WHILE END DO EXIT CYCLE
WHERE Construct	WHERE ELSEWHERE END WHERE

Order of Statements and Execution Sequence

<table>
<tr><td colspan="3">PROGRAM, FUNCTION, SUBROUTINE,
MODULE or BLOCK DATA statement</td></tr>
<tr><td colspan="3">USE statements</td></tr>
<tr><td colspan="3">IMPORT statements</td></tr>
<tr><td rowspan="4">FORMAT and ENTRY Statements</td><td colspan="2">IMPLICIT NONE</td></tr>
<tr><td>PARAMTER Statements</td><td>IMPLICIT Statements</td></tr>
<tr><td>PARAMTER and DATA Statements</td><td>Derived-type Definitions, Interface Blocks, Type Declaration Statements, Statement Function Statements and Specification Statements.</td></tr>
<tr><td>DATA Statements</td><td>Executable Constructs</td></tr>
<tr><td colspan="3">CONTAINS Statement</td></tr>
<tr><td colspan="3">Internal Subprograms or Module Subprograms</td></tr>
<tr><td colspan="3">END statement</td></tr>
</table>

Statements Allowed in Scoping Units

Scoping unit →	Main Prog	Module[3]	Block Data	External subprog	Module subprog	Internal subprog	Interface Body
USE	Yes	Yes	Yes	Yes	Yes	Yes	Yes
ENTRY	No	No	No	Yes	Yes	No	No
FORMAT	Yes	No	No	Yes	Yes	Yes	No
DATA	Yes	Yes	Yes	Yes	Yes	Yes	No
CONTAINS	Yes	Yes	No	Yes	Yes	No	No
Derived data type definition	Yes	Yes	Yes	Yes	Yes	Yes	Yes
Interface block	Yes	Yes	No	Yes	Yes	Yes	Yes
Executable statement	Yes	No	No	Yes	Yes	Yes	No
Statement function statement	Yes	No	No	Yes	Yes	Yes	No
Misc[1]	Yes	Yes	Yes	Yes	Yes	Yes	Yes

Notes

1. Miscellaneous declarations are PARAMETER statements, IMPLICIT statements, type declaration statements, and specification statements such as PUBLIC, SAVE, etc.

2. Derived type definitions are also scoping units, but they do not contain any of the above statements, and so have bot been listed in the table.

3. The scoping unit of a module does not include any module subprograms that the module contains.

Data types

Type Declaration	Conversion
INTEGER	INT(arg, kind) IDINT(arg, kind) IFIX(arg, kind)
REAL	REAL(arg, kind) FLOAT(arg, kind) SNGL(arg, kind)
CHARACTER	INTEGER:: i CHARACTER(len=10) :: ch .... WRITE(ch,*) i
COMPLEX	CMPLX(x,y, kind)

Type Declaration Statements

NON_OVERRIDABLE
Declares a bound procedure cannot be overridden in a subclass of this class.

```
PROCEDURE,NON_OVERRIDABLE::pr
```

ALLOCATABLE
Declares an array is allocatable.

```
REAL,ALLOCATABLE,DIMENSION(:)::a = -1
```

DIMENSION
Declares the rank and and shape of an array.

```
REAL,DIMENSION(-7:10,3:10)::matrix = -1
```

EXTERNAL
Declares that a name is a function external to a program unit.

```
REAL,EXTERNAL::fun1
```

INTENT
Specifies the intended use of a dummy argument.

```
REAL,INTENT(IN)::ndim
```

INTRINSIC
Declares that a name is a specific intrinsic function

```
REAL,INTRINSIC::sin
```

NOPASS
Declares a bound procedure cannot be overridden in a subclass of this class.

```
PROCEDURE,NOPASS::add
```

OPTIONAL
Declares that a dummey argument is optional.

```
REAL,OPTIONAL,INTENT(IN)::maxval
```

NON_OVERRIDABLE
Declares a bound procedure cannot be overridden in a subclass of this class.

```
PROCEDURE,NON_OVERRIDABLE::pr
```

PARAMETER
Defines named constant..

```
REAL,PARAMETER::PI=3.141593
```

PASS
Declares that the derived data type variable used to invoke a bound procedure will be passed to its as its first calling argument.

```
PROCEDURE,PASS::add
```

POINTER
Declares that a variable is a pointer.

```
INTEGER,POINTER::ptr
```

PRIVATE
Declares that an object is private to a module.

```
REAL,PRIVATE::internal_data
```

PROTECTED
Declares that an object in a module is protected, meaning that it can be used but not modified outside the module in which it is defined.

```
REAL,PROTECTED::x
```

PUBLIC
Declares that an object is private to a module.

```
REAL,PUBLIC::cir=2.54
```

SAVE
Declares that an object is private to a module.

```
REAL,SAVE::sum SAVE
```

TARGET
Declares that an object is private to a module.

```
REAL,TARGET::val1
```

VOLATILE
Declares that a value of a variable might be changed at any time by some source external to the program.

```
REAL,VOLATILE::vol1
```

Derived Data Types

Arrays

- Arrays can be up to seven dimensions.
- Fortran 90 allows the use of arithmetic array operations without the use of loops.
- Unsubscripted arrays are passed by reference. Subscripted arrays are passed by value?
- Arrays are stored in column major format. This is not the same as C which is stored in row major format.

Terminology

Automatic Arrays -
Adjustable Arrays
Assumed-shape Arrays -
Deferred-shape Arrays -
Allocatable Arrays -
Array Pointers -
Assumed-size Arrays -

Declaration

Explicit Shaped Arrays

Can have the attribute of ALLOCATABLE.
`INTEGER, DIMENSION (10) :: arr_a = -1` - A rank one array having ten elements starting at subscript 0? It is good practice to intailize your array with a value. in this case -1.

`REAL, DIMENSION(-2:9,0:5) :: arr_b =-1` - A rank two array with 12 elements in the first dimension starting at subscript -2. This is different form C where array subscripting always starts from zero.

Array Constructors

vector= (/1,2,3,4/)
vector= (/ (M,M=1,10) /) - using an implied do loop.
array=(/ ((M,M=1,10),N=1,3) /)

Array Selection

arr(i,j)	Subscript for a single value
arr(i,*)	Column i of a two dimensional array.
arr(i:k,j:l)	A 2D subarray of columns i to k, rows j to l.
arr(i:k:m)	A 1D array starting at subscript i and finishing at subscript k with a stride of m.
arr1(arr2)	The elements of Array2 subscript the elements in Array1.

Functions for Determing Array Properties

ALL(Mask,*dim*)	Determines if all values are true in Mask along dimension dim.
ANY(Mask.*dim*)	Determines if any value is true in Mask along dimension dim.
ALLOCATED(Array)	Returns true if array is allocated.
COUNT(Mask,Dim)	Returns the number of true elements in Mask along dimension Dim.
MINLOC(arr)	Returns smallest element of entire array.
MINVAL(arr,*dim*)	Returns smallest element in dimension of array.
MAXLOC(*arr*)	Returns largest element of entire array.
MAXVAL(arr,*dim*)	Returns largest element in dimension of array.
UBOUND(arr,*dim*)	a)Returns the upper bound of the subscript for the the array b)If the array argument is an array selection then result is the number of elements.
LBOUND(arr,*dim*)	a)Returns the lower bound of the subscript for the the array b)If the array argument is an array selection then result is 1.
SHAPE(arr,*dim,mask*)	Returns a one dimensional integer array. With each element being the extent of the dimensions of the source array.
SIZE(arr,*dim*)	Returns the total number of elements in the array.
SUM(arr,*dim,mask*)	Calculates the sum of selected elements in the array. Similar to total() in IDL.

Array Manipulation Functions

CSHIFT(Array,shift, *Dim*)
: Circular shift on a rank 1 array or rank 1 section of higher rank arrays.

PACK(arr,mask,vec)
: Takes some or all elements from an array and packs them into a one dimensional array, under the control of a mask.

RESHAPE(source_arr,shape,*pad,order*)
: Constructs an array of a specified shape from the elements of a given array.

TRANSPOSE(matrix)
: Takes the transpose of a 2d array (i.e matrix) turning each column into a row.

UNPACK(vec,mask,field)
: Takes some or all elements from a one dimensional array and re-arranges them into another, possibly larger array.

MERGE(Tsource,Fsource,Mask)
: Merges two arrays based on a logical mask.

EOSHIFT(Array, Shift,*Boundary,Dim*)
: End of shift of a rank 1 array or rank 1 section of a higher-rank array.

MATMUL(Matrix_1, Matrix_2)
: Performs mathematical matrix multiplication of the array arguments.

PRODUCT(arr,dim,mask)
: Multiplies together all elements in an entire array, or selected elements from all vectors along a dimension.

SPREAD(source_arr,dim,ncopies)
: Replicates an array in a additional dimension by making copies of existing elements along that dimension.

TRANSFER(source,mold,*size*) ??
: Returns either a scalar or rank 1 array with a physical representation identical to that of SOURCE, but interpreted with type and kind of MOLD. Effectively this function takes the bit patterns of SOURCE and interprets them as though they were the type and kind of MOLD.

Miscellanous Array statements

FORALL (I = 1:N, J = 1:N) H(I, J) = 3.14
Allows elements of the array to worked on in a parrallel processing environment
name: FORALL (I = 1:N, J = 1:N)
H(I, J) = 3.14
END FORALL

Structures/Derived Data Types

Unlike arrays structures allow different data types to be packaged together into one entity. They are similar to Structures in C and Derived Data types in Fortran.

Type Conversion Functions

AIMAG(Z) Imaginary part of a complex number.

AINT(R,*kind*) Returns R truncated to a whole number.

ANINT(R,*kind*) Returns the nearest whole number to R.

CEILING(R,*kind*) Returns the smallest integer greater than R.

CMPLX(X,Y)*kind*) Returns a complex value as follows. 1) If X is complex, then Y must not exist, and the value of X is returned. 2) If X is not complex, and Y does,nt exst, then the returned value is (X,0). 3) If X is not complex and Y exists, then returned value is (X,Y).

CONJG(Z) Returns the complex conjugate of a complex argument.

DBLE(A) Converts value of A to double-precision real. If A is complex, then only the real part of A is converted.

IBITS(C) ???

INT(A,*kind*) Returns a truncated A If A is complex then only the real part is converted.

LOGICAL(L,*kind*) Converts the logical value L to the specified kind.

NINT(R,*kind*) Returns the nearest integer to the real value A.

REAL(A, kind) Converts A into a real value. If A is complex, it converts the real part only.

SIGN(A,B) Returns the value of A with the sign of B.

Intrinsic Mathematical Procedures

ABS(A) Returns the absolute value of A. If complex returns $\sqrt{real^2 + imag^2}$

ACOS(X) Returns the arcosine of X.

AIMAG(Z) Returns the imaginary part of the complex argument Z.

ASIN(X) Returns the arcsine of X.

ATAN(X) Returns the arctan of X.

ATAN2(Y,X) Returns the arctan of Y/X in the range of $-\pi$ to π

COS(X) Returns the cosine of X.

COSH(X) Returns the hyperbolic cosine of X.

DIM(X,Y) Returns X-Y if > 0, otherwise returns 0. Both X and Y must be of the same type and kind.

DOT_PRODUCT(Vector_1,Vector_2)
Performs the mathematical dot product of the two rank 1 arrays.

DPROD(X,Y) Returns the double precision product of X and Y.

EXP(X) Returns e^x.

FLOOR(A,*kind*)
Returns the largest integer $\leq$ A.

LOG(X) Returns the natural logarithm of X

LOG10(X) Returns the logarithm of X to the base of 10.

MATMUL(Matrix_1, Matrix_2)
Performs mathematical matrix multiplication of the array arguments.

MAX(A1,A2,A3) Returns the maximum value of A1,A2 etc.

MIN(A1,A2,A3) Returns the minimun value of A1,A2 etc.

MOD(A,P) The remainder of A/P.

MODULO(A,P) Returns the modulo of A.

RANDOM_NUMBER(harvest)
Returns psudorandom number(s) from a uniform distribution of 0 to 1. 'harvest' may be either a scalar or an array.

RANDOM_SEED(*size*,put,*get*)
Performs three functions 1)Restarts the peudorandom number generator in RANDOM_NUMBER 2) Gets information about the generator. 3) Puts a new seed into the generator.

SIN(X) Returns the sine of X.

SINH(X) Returns the hyperbolic sine of X.

SQRT(X) Returns the square root of X.

TAN(X) Returns the tangent of X.

TANH(X) Returns the hyperbolic tangent of X.

Kind and Numeric Processor Intrinsic Functions

BIT_SIZE(I) Returns the number of bits in integer I.

DIGITS(X) Returns the number of significant digits in X in the base of the numbering system. Which is in most cases is 2. If you want the number of significant decimal digits us PRECISION(X).

EPSILON(R) Returns a positive number that is almost negligible compared to 1.0 of the same type and kind as R. R must be a real. Essentially the result is the number that when added to 1.0, produces the next number representable by the given KIND of a real number on a particular processor.

EXPONENT(X) Returns the exponent of X in the base of the the computer numbering system.

FRACTION(X) Returns the mantissa or fractional part of the model representation of X.

HUGE(X) Returns the largest number of the same type and kind as X.

KIND(X) Returns the kind value of X.

MAXEXPONENT(R)
Returns the maximium exponent of the same type and kind as R.

MINEXPONENT(R)
Returns the minimum exponent of the same type and kind as R.

NEAREST(X,S) Returns the nearest machine-representable number different from X in the direction of S. The returned value will be of the same kind as X.

PRECISION(A)
: Returns the number of significant decimal digits in values of the same type and kind as A.

RADIX(A)
: Returns the base of the mathematical model for the type and kind of I or R. Since most modern computers work on a base of 2. This number will almost certainly be 2.

RANGE(X)
: Returns the deciaml exponent range for values of the same type and kind as X.

RRSPACING(R)
: Returns the reciprocal of the relative spacing of the numbers near R.

SCALE(R,I)
: Returns the value $x * b^I$, where b is the base (Which is almost always 2).

SELECTED_CHAR_KIND(String)
: Returns the kind number associated with the character input argument.

SELECTED_INT_KIND(I)
: Returns the kind number for the smallest integer kind that can be represent al integers n whose values satisfy the condition $ABS(n) < 10 * *I$. If more than one kind satisfies this constraint, then the kind returned will be the one with the smallest decimal range. If no kind satisfies the requiremnt, the value -1 is returned.

SELECTED_REAL_KIND(P,A)
: Returns the kind number for the smallest real kind that has a decimal precison of at least P digits and an exponent range of a least A powers of 10. If more than one kind satisfies the the constraint, then the kind returned will be the one with the smallest decimal precision. If no real kind satisfies the requirement, 1) If the requested precision is not available a -1 is returned. 2) If the requested precision is available a -2 is returned. 3) If neither is available a -3 is returned. Both P and A must be integers.

SET_EXPONENT(X,I)
: Returns the number whose fractional part is the part is the fractional part of the number, and whose exponent part is I. If X is 0 the the result is 0. X must be real

SPACING()
: Returns the absolute spacing of the numbers near X in the model used to represent real numbers. If the absolute spacing is out of range, then this function returns the same value as TINY(X). The result is useful for establishing convergence criteria in a processor-indepenent manner.

TINY()
: Returns the smallest positive number of the same type and kind as X.

Intrinsic Character Functions

ACHAR(I,*kind*)
: Returns character in position I of the ASCII collating sequence.

ADJUSTL(string)
: Adjust string left, inserting trailing blanks and removing leading blanks.

ADJUSTR(string)
: Adjust string right, removing trailing blanks and inserting leading blanks.

CHAR(I,*Kind*)
: Returns character in position I of the processor collating sequence associated with the specified kind.

IACHAR(C)
: Returns the te position of the character argument in the ASCII collating sequence.

ICHAR(C)
: Returns the position of the character in the processor collating sequence.

INDEX(String,Substring,*Back*)
: Locates one substring in another, i.e returns position of Substring in characters.

LEN_TRIM(String)
: Returns the length of a character string without any trailing blank characters.

LGE(Str_a,Str_b)
: Tests whether a string is lexically greater than or equal to another string, based on the ASCII collating sequence.

LGT(Str_a,Str_b)
: Tests whether a string is lexically greater than another string, based on the ASCII collating sequence.

LLE(Str_a,Str_b)
: Tests whether a string is lexically less than or equal to another string, based on the ASCII collating sequence.

LLT(Str_a,Str_b)
: Tests whether a string is lexically less than another string, based on the ASCII collating sequence.

NEW_LINE(C)
: Returns the newline character for the KIND of the input character string.

REPEAT(Str,n_copies)
: Concatenate several copies of a string.

SCAN(Str,Set,*Back*)
: Scan a string for any one of the characters in a set of characters. Returns the position of the left most character of *str* that is in *set*.

TRIM(Str,SubStr,back)
: Returns the string without any trailing blank characters.

VERIFY(Str,Set,*Back*)
: Verify that a set of characters contains all the characters in a string. Returns the first character in the string that does NOT appear in the set.

Input/Output

OPEN(unit,file, *iostat*)
: Opens a file for I/O.There are too many options which this statement has for the space here.

READ(unit,fmt, *iostat*),var
: Reads a file in a variable. There are too many options which this statement has for the space here.

WRITE(unit,fmt,*iostat*),var
: Writes a variable to a file. There are too many options which this statement has for the space here.

CLOSE(unit,iostat,err,status)
: Closes a particular file unit.

FLUSH(unit)
: Flush output buffers to disk.

WAIT(unit)
: Wait for asynchronous I/O to complete.

UNIT=5 for stdin,
and UNIT=6 for stdout

Pointers

POINTER
: Attribute must be used in variable declaration.

TARGET
: Attribute must be used in variable declaration.

var_1 => var_2
: Assigns the pointer from variable 1 to variable 2.

ASSOCIATED(var_1)
: Returns a logical result depending on whether the pointer has been associated.

NULL(MOLD)??
: Returns a disassociated pointer of the same type as MOLD if present. If MOLD is not present, the pointer type is determined by context. MOLD is a pointer of any type. Its pointer association status may be undefined, disassociated, or associated. This function is useful for initializing the status of a pointer at the time it is declared.

NULLIFY(var_1)
: Causes pointer to become disassociated. If the pointer is not assigned to anything it is good programming practice to have them disassociated. Always initialize as pointer iwth NULLIY or with the pointer assigned

ALLOCATE(var_1)
: Dynamically provides storage for pointer targets and allocatable arrays.

Miscellaneous Functions

PRESENT(A)??? Returns true if optional argument A is present.

Debugging techniques

1. Switch on all error testing that can be provided by the compiler.
2. Use interface blocks to trap a very common error which is parameter mismatch between calling and called subroutine.
3. Check for mixed-mode arithmetic.
4. Putting in simple print statements.

Good programming Practise

1. Use meaningful variable names.
2. Use IMPLICIT NONE.
3. Echo all input values.
4. Create a data dictionary in each program that you write. Including the physical units used.
5. Specify constants with a much precision as your computer will support.
6. Initialize all variables.
7. Always print the physical units associated with any value.

Useful Links

`www.fortran.com`

`comp.lang.fortran` - Usenet group.
