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 executable 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, 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) Skips a line in a fmt statement?
; (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 character. (Must be in line 7 of fixed format F77. For F90 can be anywhere after the line.

Concepts and Elements

<table>
<thead>
<tr>
<th>Concept</th>
<th>Statements</th>
</tr>
</thead>
<tbody>
<tr>
<td>Module</td>
<td>Contains</td>
</tr>
<tr>
<td></td>
<td>Private</td>
</tr>
<tr>
<td></td>
<td>Public</td>
</tr>
<tr>
<td></td>
<td>End Module</td>
</tr>
<tr>
<td>Interface Block</td>
<td>Interface</td>
</tr>
<tr>
<td></td>
<td>Module Procedure</td>
</tr>
<tr>
<td></td>
<td>End Interface</td>
</tr>
<tr>
<td>Derived data type</td>
<td>Derived type</td>
</tr>
<tr>
<td></td>
<td>Private</td>
</tr>
<tr>
<td></td>
<td>Sequence</td>
</tr>
<tr>
<td></td>
<td>End Type</td>
</tr>
<tr>
<td>Subprogram</td>
<td>Function</td>
</tr>
<tr>
<td></td>
<td>Subroutine</td>
</tr>
<tr>
<td></td>
<td>Entry</td>
</tr>
<tr>
<td></td>
<td>Contains</td>
</tr>
<tr>
<td></td>
<td>Return</td>
</tr>
<tr>
<td>Input/Output</td>
<td>Backspace</td>
</tr>
<tr>
<td></td>
<td>Close</td>
</tr>
<tr>
<td></td>
<td>Endfile</td>
</tr>
<tr>
<td></td>
<td>Format</td>
</tr>
<tr>
<td></td>
<td>Inquire</td>
</tr>
<tr>
<td></td>
<td>Open</td>
</tr>
<tr>
<td></td>
<td>Print</td>
</tr>
<tr>
<td></td>
<td>Read</td>
</tr>
<tr>
<td></td>
<td>Rewind</td>
</tr>
<tr>
<td></td>
<td>Write</td>
</tr>
</tbody>
</table>

Flow Control

<table>
<thead>
<tr>
<th>Group</th>
<th>Statements</th>
</tr>
</thead>
<tbody>
<tr>
<td>IF</td>
<td>IF</td>
</tr>
<tr>
<td></td>
<td>ELSE IF</td>
</tr>
<tr>
<td></td>
<td>ELSE</td>
</tr>
<tr>
<td></td>
<td>ENDIF</td>
</tr>
<tr>
<td>CASE</td>
<td>SELECT CASE</td>
</tr>
<tr>
<td></td>
<td>CASE</td>
</tr>
<tr>
<td></td>
<td>END SELECT</td>
</tr>
<tr>
<td>Do/Do while</td>
<td>DO</td>
</tr>
<tr>
<td></td>
<td>DO WHILE</td>
</tr>
<tr>
<td></td>
<td>END DO</td>
</tr>
<tr>
<td></td>
<td>EXIT</td>
</tr>
<tr>
<td></td>
<td>CYCLE</td>
</tr>
<tr>
<td>WHERE Construct</td>
<td>WHERE</td>
</tr>
<tr>
<td></td>
<td>ELSEWHERE</td>
</tr>
<tr>
<td></td>
<td>END WHERE</td>
</tr>
</tbody>
</table>

Order of Statements and Execution Sequence

<table>
<thead>
<tr>
<th>Scoping unit →</th>
<th>Main Module</th>
<th>Block External Module</th>
<th>Internal Interface BODY</th>
<th>Internal Interface Prog</th>
<th>Internal Data subprog</th>
<th>Block Data subprog</th>
<th>Block subprog</th>
<th>Body</th>
</tr>
</thead>
<tbody>
<tr>
<td>USE</td>
<td>Yes</td>
<td>Yes</td>
<td>Yes</td>
<td>Yes</td>
<td>Yes</td>
<td>Yes</td>
<td>Yes</td>
<td>Yes</td>
</tr>
<tr>
<td>ENTRY</td>
<td>No</td>
<td>No</td>
<td>No</td>
<td>Yes</td>
<td>Yes</td>
<td>Yes</td>
<td>No</td>
<td>No</td>
</tr>
<tr>
<td>FORMAT</td>
<td>Yes</td>
<td>No</td>
<td>No</td>
<td>Yes</td>
<td>Yes</td>
<td>Yes</td>
<td>Yes</td>
<td>No</td>
</tr>
<tr>
<td>DATA</td>
<td>Yes</td>
<td>Yes</td>
<td>Yes</td>
<td>Yes</td>
<td>Yes</td>
<td>Yes</td>
<td>Yes</td>
<td>No</td>
</tr>
<tr>
<td>CONTAINS</td>
<td>Yes</td>
<td>Yes</td>
<td>No</td>
<td>Yes</td>
<td>Yes</td>
<td>Yes</td>
<td>No</td>
<td>No</td>
</tr>
<tr>
<td>Derived data type definition</td>
<td>Yes</td>
<td>Yes</td>
<td>No</td>
<td>Yes</td>
<td>Yes</td>
<td>Yes</td>
<td>Yes</td>
<td>Yes</td>
</tr>
<tr>
<td>Interface block</td>
<td>Yes</td>
<td>Yes</td>
<td>No</td>
<td>Yes</td>
<td>Yes</td>
<td>Yes</td>
<td>Yes</td>
<td>Yes</td>
</tr>
<tr>
<td>Executable statement</td>
<td>Yes</td>
<td>No</td>
<td>No</td>
<td>Yes</td>
<td>Yes</td>
<td>Yes</td>
<td>No</td>
<td>No</td>
</tr>
<tr>
<td>Statement function statement</td>
<td>Yes</td>
<td>No</td>
<td>No</td>
<td>Yes</td>
<td>Yes</td>
<td>Yes</td>
<td>No</td>
<td>No</td>
</tr>
</tbody>
</table>

Statements Allowed in Scoping Units

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 not been listed in the table.

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

<table>
<thead>
<tr>
<th>Type Declaration</th>
<th>Conversion</th>
</tr>
</thead>
<tbody>
<tr>
<td>INTEGER</td>
<td>INT(arg, kind)</td>
</tr>
<tr>
<td></td>
<td>IDINT(arg, kind)</td>
</tr>
<tr>
<td></td>
<td>IFIX(arg, kind)</td>
</tr>
<tr>
<td>REAL</td>
<td>REAL(arg, kind)</td>
</tr>
<tr>
<td></td>
<td>FLOAT(arg, kind)</td>
</tr>
<tr>
<td></td>
<td>SNGL(arg, kind)</td>
</tr>
<tr>
<td>CHARACTER</td>
<td>INTEGER:: i</td>
</tr>
<tr>
<td></td>
<td>CHARACTER(len=10) :: ch</td>
</tr>
<tr>
<td></td>
<td>WRITE(ch,* ) i</td>
</tr>
<tr>
<td>COMPLEX</td>
<td>CMPLX(x,y, kind)</td>
</tr>
</tbody>
</table>

**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 dummy 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

**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 initialize your array with a value. in this case -1.

**Functions for Determining 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 in dimension of array.
- **MAXLOC(arr)**
  - 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 Constructors**

```fortran
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**

```fortran
arr(i,j)  
arr(i,*)  
arr(i:k,j:l)  
arr(i:k:m)  
arr1(arr2)  
```

**Example**

```fortran
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 Manipulation Functions

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

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.

EDSHIFT(Array, Shift, Boundary, Dim)
End of shift of a rank 1 array or rank 1 section of a higher rank array.

MATMUL(Matrix1, Matrix2)
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, size)
Replicates an array in a additional dimension by making copies of existing elements along that dimension.

Miscellaneous Array statements

FORALL (I = 1:N, J = 1:N) H(I, J) = 3.14
Allows elements of the array to worked on in a parallel 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 not exist, 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(A)
Returns a truncated A If A is complex, then only the real part of A is converted.

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

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

MATMUL(Matrix1, Matrix2)
Performs mathematical matrix multiplication of the array arguments.

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

MIN(A1, A2, A3)
Returns the minimum value of A1, A2 and A3.

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

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

RANDOM_NUMBER(harvest)
Returns pseudorandom 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 pseudorandom 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.

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 use 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.

EXPTION(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 maximum 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.

Intrinsic Mathematical Procedures

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

ACOS(X)
Returns the arccosine 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 \(\pi\)

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

COS(X)
Returns the cosine of X.

COSH(X)
Returns the hyperbolic cosine of X.

DIM(X, Y)
Returns X-Y if type and kind as X.

DOT_PRODUCT(Vector1, Vector2)
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.
Intrinsic Character Functions

- **ACCHAR(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.
- **ICHAR(C)** Returns the te position of the character argument in the ASCII collating sequence.
- **INDEX(String, Substring, Back)** Returns the position of the character in the processor collating sequence.
- **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(Tr, Str_b)** Tests whether a string is lexically greater than another string, based on the ASCII collating sequence.
- **LLE(Tr_a, Str_b)** Tests whether a string is lexically less than or equal to another string, based on the ASCII collating sequence.
- **LLT(Tr_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(Tr, n_copies)** Concatenate several copies of a string.
- **SCAN(Tr, 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(Tr, SubStr, back)** Returns the string without any trailing blank characters.
- **VERIFY(Tr, 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.

Pointers

- **POINTER** Attribute must be used in variable declaration.
- **TARGET** Attribute must be used in variable declaration.

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.

This card was created using \LaTeX. Released under the GNU general public license. $\text{Revision: 0.118 \$, \text{Date: 27/02/2009 \$. To contact me regarding improvements/mistakes on this sheet or to download the latest version please follow the links from: http://www.BenjaminEvans.net}$