Fortran 90/95 syntax

Program structure

```
PROGRAM <Program Name>
   IMPLICIT NONE
        !parameters
        <Variable Type>,parameter:: <Parameter Name>=<Value>
   !Program variable definition
        <Variable Type>:: <Variable Name> [= <Default value>],<Variable Name>
        INTEGER[*2] :: <Variable Name> [= <Default value>]
        REAL[*8]:: <Variable Name> [= <Default value>]
        Character::<variable Name>*<size>
        Logical:: <variable Name> [=.true./.False.]

        !arrays
        <Variable Type>:: <Variable Name>(<Array Size>)!one dimensions
        <Variable Type>:: <Variable Name>(<From>:<to>)!one dimensions
        <Variable Type>:: <Variable Name>(<Array Size>,<Array Size>)!two
        <Variable Type>:: <Variable Name>(<From>:<to>,<From>:<to>)!two
        <Variable Type>,allocatable:: <Variable Name>(:)
        !program body

        STOP
CONTAINS
        SUBROUTINE <Name>(<in/out param>,<in/out param>)
        !in/out Param definition
        <Param Type>, INTENT(<in/out/in out>)[,OPTIONAL]::<Param Name>
        INTEGER, INTENT(IN) :: param1
        !arrays and chars
         <variable type>, intent(<in/out/in out>) :: <array name>(:[,:])
         character,intent(<in/out>) :: <string name>*(*)

        !internal variables like in program
        !Subroutine body

        Return
        END SUBROUTINE [<Name>]

        Function <Name>(<in/out param>,<in/out param>)
        <Param Type>::<Function Name>
        !in/out Param definition like Subroutine
        !internal variables like in program

        !Function body
        <Function Name> = <Return value>
        Return
        END Function <Name>

        Recursive Function <Name>(<param>,<param>) result(<Param Name>)
        <Param Type>::<Function Name>
        !in/out Param definition like Subroutine
        !internal variables like in program

        !Function body
        <Param Name> = <Return value>
        Return
        END Function <Name>
END PROGRAM <Program Name>
```

Operators

`**`		exponentiation
`*`		multiplication
`/`		division
`+`		addition
`-`		subtraction
`//`		concatenation
`==`	`.eq.`	equality
`/=`	`.ne.`	not equal
`<`	`.lt.`	less than
`>`	`.gt.`	greater than
`<=`	`.le.`	less than or equal
`>=`	`.ge.`	greater than or equal
`.not.`		complement, negation
`.and.`		logical and
`.or.`		logical or

SELECT CASE

```
  SELECT CASE(<Variable Name>) !only integer, Logic & Character
  CASE(<Value1>,<Value2>,<Value3>)
        !executable statements
  CASE(<from>:<to>,<from>:<to>)
        !executable statements
  CASE DEFAULT
        !executable statements
  END SELECT
```

Do

```
DO <integer Variable> = <start Value>,<End Value>[,<Step value>]
!executable statements
        IF(<condition>) EXIT
        IF(<condition>) CYCLE

END DO

!An indefinite DO also exists - here an EXIT from the loop is essential:
DO
        !executable statements
        IF(<condition>) EXIT
        IF(<condition>) CYCLE

END DO

DO WHILE(<condition>)
        !executable statements
        IF(<condition>) EXIT
        IF(<condition>) CYCLE

END DO
```

IF

```
IF (<condition>) THEN
        !executable statements
        ELSE IF (<condition>) then
        !executable statements
                ELSE
                !executable statements
END IF
```

I/O

```
Read(<5/*/File Number>,<*/Format/Format Number> [,end=<Number>]
[,advance='no'] [,eor=<Number>]) <variable>,<value>

write(<6/*/File Number>,<*/Format/Format Number>[,advance='no'])
<variable>,<value> ! advance='no' requires format

Write/read (<char name>,<format/format num>) <variable/value>


Open(<File Number>,file='<File Name>')
Close(<File Number>)

Rewind(<File Number>)

<Format Number> Format(<Format String>)
```

Formats

Descriptor	Description
*R*lw	Edits integer data
*r*Fw.d	Edits both real and double precision data in decimal format
*r*Ew.d	Edits real data in exponential format
*r*Dw.d	Edits double precision data in exponential format
*r*Gw.d	Edits both real and double precision data in exponential format
Lw	Edits logical data
*r*Aw	Edits character data
'a..a'	Specifies a character constant
Tc	Tabs to position c
TLn	Tabs backward n positions
TRn	Tabs forward n positions
nX	Skips over n positions (same as TRn)
/	Causes the current record to be written or the next record to be read
SS	Suppresses printing of plus sign
BN	Ignores blank spaces in a field
BZ	Considers blank spaces in a field to be zeros
*k*P	Multiplies each number by 10-k on input and 10kon output. The scale factor must precede an E,F,D, or G descriptor
N(any)	Repeat n times

Notes:
r is an optional unsigned nonzero positive integer used as repeat count.
w is an unsigned nonzero positive integer that specifies the data field width.
d is an unsigned positive integer that specifies the number of places to the right of the decimal point.
a is any character.
c is an unsigned positive nonzero integer.
n is an unsigned positive nonzero integer.
k is an unsigned positive nonzero integer

examples:

```
Format(tr2,i3,2f6.1,'OferFridman',tr3,i2)
```

Other

```
Call <SUBROUTINE Name>
```

Function

Function	Returns
abs(integer_real_complex) result(integer_real_complex)	Absolute value.
achar(integer [,kind=]) result(character)	Character in position I of the processor collating sequence.
adjustl(character) result(character)	left adjust, blanks go to back
adjustr(character) result(character)	right adjust, blanks to front
Allocate(array(<from>:<to>[,:]),stat=<int varible>)	Allocate array
allocated(array) result(logical)	true if array is allocated in memory
ceiling(real) result(real)	Least integer greater than or equal to its argument.
cos(real_complex) result(real_complex)	Cosine function.
dble(integer_real_complex) result(real_kind_double)	Convert to double precision real.
Deallocate	De allocate array
exp(real_complex) result(real_complex)	Exponential function.
floor(real) result(real)	Greatest integer less than or equal to its argument.
fraction(real) result(real)	Fractional part of the model for X.
IACHAR(C) result(integer)	Returns the position of a character in the ASCII collating sequence.
index(string,substring [,back=])	Starting position of SUBSTRING within STRING.
int(integer_real_complex) result(integer)	Convert to integer type.
len(character) result(integer)	Character length.
len_trim(character) result(integer)	Length of STRING without trailing blanks.
LOG (X)	Natural (base e logarithm function.
LOG10 (X)	Common (base 10) logarithm function.
MATMUL (MATRIX_A, MATRIX_B)	Matrix multiplication.
MAX (A1,A2[,A3...])	Maximum value.
MAXEXPONENT (X)	Maximum exponent in the model for numbers like X.
MAXLOC (ARRAY [,MASK]) result(array)	Location of maximum array element.
MAXVAL (ARRAY [,DIM] [,MASK])	Value of maximum array element.
MIN (A1,A2 [,A3,...])	Minimum value.
MINLOC (ARRAY [,MASK]) result(array)	Location of minimum array element.
MINVAL (ARRAY [,DIM] [,MASK])	Value of minimum array element.
MOD (A,P)	Remainder modulo P, that is A-INT(A/P)*P.
PRESENT(<VARIBLE>) result(logical)	

REAL (A [,KIND])	Convert to real type.
REPEAT (STRING, NCOPIES)	Concatenates NCOPIES of STRING.
RESHAPE (source, shape [, order])	Constructs an array with a different shape from the argument array
SIN (X)	Sine function.
SIZE (ARRAY [,DIM])	Array size
SQRT (X)	Square root function.
SUM (ARRAY, [,DIM] [,MASK])	Sum of array elements.
TAN (X)	Tangent function.
TRIM (STRING)	Remove trailing blanks from a single string.

Edited by Ofer Fridman
If you have any comments please contact me at oferfrid@hotmail.com