

FORTRAN Format

FORTRAN formats are used to control the appearance of printed output. It has the following form:

(.... format edit descriptors ...)

There are three different ways of using format:

```
CHARACTER(LEN=20), PARAMETER :: FMT1 = "(I5,F10.2)"
CHARACTER(LEN=*), PARAMETER  :: FMT2 = "(4I5, 5E14.7, 8F5.0)"
CHARACTER(LEN=80)           :: STRING
```

```
READ(*, '(...)' ) .....      ! Form #1
WRITE(*, '(...)' ) .....
```

```
READ(*, FMT1) .....          ! Form #2
WRITE(*, FMT1) .....
READ(*, FMT2) .....
WRITE(*, FMT2) .....
```

```
STRING = "(3I5, 10F8.2)"
READ(*, STRING) .....       ! Form #3
WRITE(*, STRING) .....
```

```
Form #4:
lable format(3I5,10F8.2)
read(*,lable) ...
write(*,lable) ...
```

Format Edit Descriptors

There are several commonly used edit descriptors, where w – number of positions, m – minimum number of positions, d – number of digits to the right of the decimal point, and e – number of digits in the exponential part

1. For reading/writing **INTEGERS**: Iw and $Iw.m$;

2. For reading/writing **REALS**:

(a) Decimals: $Fw.d$

(b) Scientific notation: $Ew.d$ and $Ew.dEe$

3. For reading/writing **CHARACTERS**: A and Aw

4. For reading/writing **LOGICALS**: Lw

5. For horizontal spacing: nX

6. For vertical spacing: $/$

7. For tabbing: Tc , TLc and TRc

Edit descriptors are separated by commas:

```
CHARACTER(LEN=30) :: FRMT
FRMT = "(5X, I5.2, F10.3, A)"
WRITE(*,FRMT) .....
READ(*,FRMT) .....
```

Integer Output

Editing Descriptor(s): `rIw` and `rIw.m`

- **I**: INTEGER output
- *w*: width of field (*i.e.*, the number of positions)
- *m*: minimum number of digits to be printed
- *r*: repetition indicator (`3I5.2` is equivalent to `I5.2, I5.2, I5.2`)

```
INTEGER :: A, B
```

```
A = 123
```

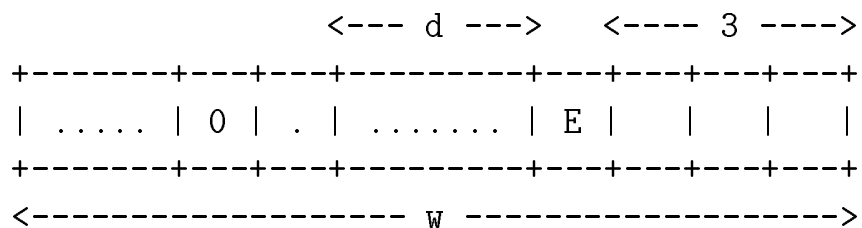
```
B = -123
```

		1	2	3	4	5
		+	-	+	-	+
WRITE(*, '(I5)') A				1		2
WRITE(*, '(I5.4)') A				0		1
WRITE(*, '(I5.5)') A			0		0	
WRITE(*, '(I5.2)') A				1		2
WRITE(*, '(I5.2)') 123456			*		*	
WRITE(*, '(I5)') B				-		1
WRITE(*, '(I5.4)') B			-		0	
WRITE(*, '(I5.5)') B			*		*	
WRITE(*, '(I5.2)') B				-		1
		+	-	+	-	+

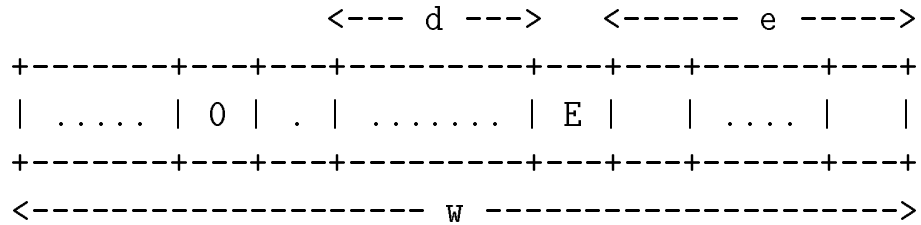
Real Output – the E Descriptor

Editing Descriptor(s): $rEw.d$ and $rEw.dEe$

- E: REAL output
- w : width of field (*i.e.*, the number of positions)
- d : number of digits to the right of the decimal point
- e : number of positions for exponent
- r : repetition indicator ($3E5.2$ is equivalent to $E5.2$, $E5.2$, $E5.2$)
- **IMPORTANT:** Before printing, the number to be printed is converted to $\pm 0.dddd \dots \times 10^n$. The actual printout is $\pm 0.dddd \dots \times 10^n$ rather than the original.
- For $Ew.d$, the w positions are arranged as follows. Thus, $w \geq d + 7$

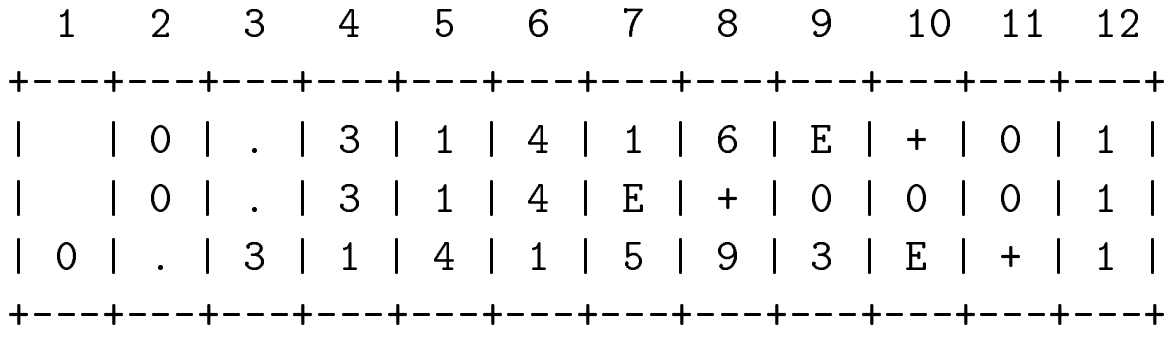


- For $Ew.dEe$, the w positions are arranged as follows.
Thus, $w \geq d + e + 5$.



```
REAL :: PI = 3.1415926
```

```
WRITE(*, '(E12.5)') PI
WRITE(*, '(E12.3E4)') PI
WRITE(*, '(E12.7E1)') PI
```



Logical Output

Editing Descriptor(s): rLw

- L: LOGICAL output
- w : width of field (*i.e.*, the number of positions)
- r : repetition indicator (3L5 is equivalent to L5, L5, L5)
- **IMPORTANT:** the output of .TRUE. and .FALSE. are **T** and **F**, respectively, and are *right justified*.

```
LOGICAL :: A, B
A = .TRUE.
B = .FALSE.
WRITE(*, '(L1,L2)') A, B
WRITE(*, '(L3,L4)') A, B
```

	1	2	3				
	+-----+-----+-----+						
	T		F				
	+-----+-----+-----+-----+-----+-----+						
			T			F	
	+-----+-----+-----+-----+-----+-----+						
	1	2	3	4	5	6	7

Spacing and Tabbing

Editing Descriptor(s): nX and Tc , TLc and TRc

- nX : insert n spaces
- Tc : move to position c
- TLc : move backward c positions
- TRc : move forward c positions
- **IMPORTANT:**
 1. No repetition can be used
 2. Nothing will be printed
 3. However, the position for printing “next” data item will be affected.

```
INTEGER :: A, B
```

```
A = 123
```

```
B = 456
```

```
WRITE(*, '(2X,I4,3X,I3)') A, B
```

```
  1  2  3  4  5  6  7  8  9  10 11 12
+---+---+---+---+---+---+---+---+---+---+---+---+
|   |   |   | 1 | 2 | 3 |   |   |   | 4 | 5 | 6 |
+---+---+---+---+---+---+---+---+---+---+---+---+
<--2X----><-----I4-----><--3X-----><----I3---->
```

```
INTEGER :: A, B
```

```
A = 123
```

```
B = 456
```

```
WRITE(*, '(T6,I4,T2,I4)') A, B
```

```
  1   2   3   4   5   6   7   8   9
+---+---+---+---+---+---+---+---+---+
|   |   | 4 | 5 | 6 |   | 1 | 2 | 3 |
+---+---+---+---+---+---+---+---+---+
```

```
INTEGER :: A, B, C
```

```
A = 123
```

```
B = 456
```

```
C = 789
```

```
WRITE(*, '(T10,I3,TL9,I3,TR5,I3)') A, B, C
```

```
          1   1   2
      ....5....0....5....0
T10          |
I3          123*
TL9         |
I3         456* 123
TR5          |
I3         456 12789* <--- the result you see
```

ignor Printer Control

This is necessary **ONLY IF** you want to print on a printer:

<i>First Char on a Line</i>	<i>Effect</i>
space	Advance to next line <u>BEFORE</u> printing
0	Advance two lines <u>BEFORE</u> printing
1	Advance to next page <u>BEFORE</u> printing
+	No advancing, print on the current line (overprinting)

Watch out the **FIRST** Character You Print

```
INTEGER :: A = 10, B = 100
```

```
1234567890
WRITE(*, '(I3)') A    10 <----- first space for advancing

1234567890
WRITE(*, '(I3)') B    00 <----- first line on next page
                      1 in 100 for form feed
```


The Grouping Descriptor: $r(\dots)$

1. The descriptors within $()$ are repeated r times.

$(1X, 3(I5, F5.2), A)$

$3(I5, F5.2)$ means $I5, F5.2$ is repeated three times.
Thus, the above is equivalent to

$(1X, I5, F5.2, I5, F5.2, I5, F5.2, A)$

2. Grouping can be nested:

$(1X, 3(I5, 2(I4, A3)), A)$

The above is equivalent to

$(1X, I5, 2(I4, A3),$
 $I5, 2(I4, A3),$
 $I5, 2(I4, A3), A)$

which in turn is equivalent to

$(1X, I5, I4, A3, I4, A3,$
 $I5, I4, A3, I4, A3,$
 $I5, I4, A3, I4, A3, A)$

3. A group with repetition factor $r = 1$ has special meaning.

Example 1

Computing means revisited.

```
PROGRAM Means
  IMPLICIT NONE
  REAL                :: a, b, c
  REAL                :: Am, Gm, Hm
  CHARACTER(LEN=*) , PARAMETER :: FMT = "(1X, A//3(1X, A, F15.7/))"

  READ(*,*) a, b, c
  Am = (a + b + c)/3.0
  Gm = (a * b * c)**(1.0/3.0)
  Hm = 3.0 / (1.0/a + 1.0/b + 1.0/c)
  WRITE(*,FMT) "Input Data", &
              "a = ", a, &
              "b = ", b, &
              "c = ", c

  WRITE(*,FMT) "Computed Results", &
              "Arithmetic Mean = ", Am, &
              "Geometric Mean = ", Gm, &
              "Harmonic Mean = ", Hm

END PROGRAM Means
```

Input Data

```
a =      1.0000000
b =      2.0000000
c =      3.0000000
```

Computed Results

```
Arithmetic Mean =      2.0000000
Geometric Mean  =      1.8171207
Harmonic Mean   =      1.6363636
```

Matching Data Values with Edit Descriptors

1. When a **WRITE** starts, its format is scanned from left to right.
2. Edit descriptors are processed until a descriptor that requires a data value (*i.e.*, **I**, **F**, **E**, **L** and **A**) is encountered.
3. Take next data item and use the encountered edit descriptor for printing the value.
4. The type of the value and the edit descriptor **MUST** match: **I** for **INTEGER**, **F** and **E** for **REAL**, **L** for **LOGICAL** and **A** for **CHARACTER**.
5. After processing a value, go back to step 2.
6. This process will continue until encounters the right parenthesis.

Matching Data Values with Edit Descriptors

If the right-most right parenthesis is reached,

1. If there is no data items left, we are done!
2. If there are un-processed data items, then the format edit descriptors will be re-scanned:
 - (a) If there is no grouping parenthesis in a format, then start a new line and rescan the format from the very beginning:

وقتی تعداد متغیرها بیشتر از فرمت باشد

```
INTEGER :: A = 10, B = 20, C = 30  
INTEGER :: D = -10, E = -20, F = -30
```

```
WRITE(*, '(1X, 2I4)') A, B, C, D, E, F
```

Output

```
-----  
10 20  
30 -10  
-20 -30
```


- (b) If there are grouping parenthesis, then find the **rightmost** right parenthesis in the format and its matching left parenthesis and start rescan there. Note that any repetition preceding this group is in effect.

```
INTEGER, DIMENSION(1:17) :: A
INTEGER                      :: i

DO i = 1, 17
    A(i) = i
END DO
WRITE(*, '(1X, 2(I3,I4), I5)') (A(i), i=1,17)
```

Output:

```
-----
                1    1    2
12345678901234567890
  1    2    3    4    5
  6    7    8    9   10
11   12 13   14   15
16   17
```

- (c) Continue this process until all data items are printed.

Example 3

Print an array on a single column

```
PROGRAM Single_Column
  IMPLICIT NONE
  INTEGER, PARAMETER          :: MAX_SIZE = 20
  INTEGER, DIMENSION(1:MAX_SIZE) :: x
  INTEGER                     :: i
  CHARACTER(LEN=80)           :: FMT

  DO i = 1, MAX_SIZE
    X(i) = MOD(i, 5) + i
  END DO

  FMT = "(T7, A/ T7, A// T10, A, T15, A/(T10, I2, T15, I4))"
  WRITE(*,FMT) 'Generated Table', &
    '-----', &
    'No', 'Data', &
    (x(i), i = 1, MAX_SIZE)
END PROGRAM Single_Column
```

Generated Table

No	Data
2	4
6	8
5	7
9	11
13	10
12	14
16	18
15	17
19	21
23	20

Example 4

Print an array six elements per row.

```
PROGRAM Six_Per_Row
  IMPLICIT NONE
  INTEGER, PARAMETER          :: SIZE = 20
  INTEGER, DIMENSION(1:SIZE) :: x
  INTEGER                     :: i
  CHARACTER(LEN=80)           :: String

  String = "(1X, A/ 1X, A // (1X, 6I4))"

  DO i = 1, SIZE
    x(i) = MOD(i, 5) + i
  END DO

  WRITE(*,String) 'Generated Table', &
    '-----', &
    (x(i), i = 1, SIZE)

END PROGRAM Six_Per_Row
```

Generated Table

```
  2  4  6  8  5  7
  9 11 13 10 12 14
 16 18 15 17 19 21
 23 20
```

Integer Input

Editing Descriptor(s): rIw

- I: INTEGER input
- w : width of field (*i.e.*, the number of positions)
- m : minimum number of digits to be printed
- r : repetition indicator (3I5.2 is equivalent to I5.2, I5.2, I5.2)
- The content in the next w positions is considered to be an integer whose value goes into the corresponding variable.
- All blanks in the field of w positions are **IGNORED** by default.

```
INTEGER :: A, B, C, D                                1    1    2
                                                    12345678901234567890
READ(*, '(4I4)') A, B, C, D                        1 2  34 56  789
                                                    <--><--><--><-->
                                                    A   B   C   D
```

A, B, C and D receives 12, 34, 56 and 789, respectively.

Real Input

Editing Descriptor(s): $rFw.d$

- F: REAL input
- w : width of field (*i.e.*, the number of positions)
- d : number of digits to the right of the decimal point
- r : repetition indicator (3F5.2 is equivalent to F5.2, F5.2, F5.2)

• Three cases are possible:

1. If there is no decimal point in the field, the rightmost d positions are considered to the right of the decimal point:

```
REAL :: A, B, C
                                1
                                123456789012345
READ(*, '(3F5.2)') A, B, C    123 456 789 012
                                <----><----><---->
                                A    B    C
```

The first field is 1234, the second is 5678 and the third is 9012. Taking the last two digits to be the fraction part, A, B and C are 12.34, 56.78 and 90.12.

2. If the input has a decimal point, the actual value will be taken and the d in $Fw.d$ has no effect.

```

REAL :: A, B, C
                                1    1    2
                                123456789012345678901
READ(*, '(3F7.2)') A, B, C 12.3   -1.45  99.768
                                <-----><-----><----->
                                A      B      C

```

A, B and C receives 12.3, -1.45 and 99.768.

- You can use scientific notation by adding an E followed by an exponent. The remaining part is handled by the above two rules based on if a decimal point presents.

```

REAL :: A, B,
                                1    1    2
                                12345678901234567890
READ(*, '(3F10.2)') A, B  1 3 5 E+2  123.3E-1
                                <-----><----->
                                A      B

```

The field for A, after removing spaces, is 135E+2. Since the edit descriptor is F10.2, the 135 part is read as 1.35 (two decimal places) and the actual value is 1.35×10^2 . Thus, A receives 135.0 and B has $123.3 \times 10^{-1} = 12.33$.

Logical Input

Editing Descriptor(s): *rLw*

- **L**: **LOGICAL** input
- *w*: width of field (*i.e.*, the number of positions)
- *r*: repetition indicator (**3L5** is equivalent to **L5**, **L5**, **L5**)
- The *leftmost* **T** or **F** in the field yields the value of **.TRUE.** or **.FALSE.**

```
LOGICAL :: A, B, C, D                1    1    2
                                     12345678901234567890
READ(*, '(4L5)') A, B, C, D         T    F  ABT  XYFT
```

A and **B** receive **.TRUE.** and **.FALSE.** For **C**, its input field consists of a blank, character **A**, character **B** and character **T**. Thus, **C** receives **.TRUE.** Similarly, **D** receives **.FALSE.**

Character Input

Editing Descriptor(s): rA and rAw

- A : CHARACTER input
- w : width of field (*i.e.*, the number of positions)
- r : repetition indicator ($3A5$ is equivalent to $A5$, $A5$, $A5$)
- **IMPORTANT:**
 1. If w is missing, the length of the CHARACTER variable is used.
 2. If w is larger than the length of the CHARACTER variable, only the *rightmost* part is taken and put into the variable.
 3. If w is smaller than the length of the CHARACTER variable, spaces will be added and put into the variable.

```
CHARACTER(LEN=5) :: A, B, C           123456
READ(*, '(A4)')  A                     LMNOP
READ(*, '(A5)')  B                     LMNOP
READ(*, '(A6)')  C                     LMNOPQ
```

A receives 'LMNO ', B receives 'LMNOP' and C receives 'MNOPQ'.

Spacing and Tabbing

Editing Descriptor(s): nX and Tc . Note that no repetition can be used.

- nX : insert n spaces
- Tc : move to position c

```
INTEGER :: A, B, C, D
```

```
READ(*, '(2X,I3,T7,I4,3X,I1,T18,I2)') A, B, C, D
```

```
Input:      12345678901234567890
              |
2X           |
I3           345| (A receives 345)
T7           |
I4           7890| (B receives 7890)
3X           |
I1           4| (C receives 4)
T18          |
I2           89 (D receives 89)
```

The Slash (/) Descriptor

1. Advance to next input line. The remaining input on the current line will be **IGNORED**.
2. Between two adjacent /s, no comma is required. // is the same as /, /.

```
INTEGER :: A, B, C, D
```

```
READ(*, '(I2/T5,I3//I4/I5)') A, B, C, D
```

```
Input:  12345678 <--- I2 (A receives 12)      /
        90123456 <--- T5, I3 (B receives 345) /
        78901234 <---                          /
        56789012 <--- I4 (C receives 5678)    /
        34567890 <--- I5 (D receives 34567)
        ..... <--- next READ starts here
```

