FORTRAN Format

FORTRAN formats are used to control the appearance of printed output. Its 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) .....
```

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 `INTEGER`s: `I`w and `I`$w.m$;
2. For reading/writing `REAL`s:
   (a) Decimals: `F`$w.d$
   (b) Scientific notation: `E`$w.d$ and `E`$w.d$`E`e
3. For reading/writing `CHARACTER`s: `A` and `A`w
4. For reading/writing `LOGICAL`s: `L`w
5. For horizontal spacing: n`X`
6. For vertical spacing: `/`
7. For tabbing: `T`c, `TL`c and `TR`c

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): r`I`w and r`I`$w.m$

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

Real Output – the F Descriptor

Editing Descriptor(s): $r\texttt{F}w.d$

- `F`: `REAL` output
- 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`)
- **<u>IMPORTANT</u>**: the integer part has only $w \geq d+2$ positions to use (1 for decimal point and 1 for sign)

```
REAL :: A, B
A = 123.345                1   2   3   4   5   6   7   8   9   10
                         +---+---+---+---+---+---+---+---+---+---+
WRITE(*, '(F10.0)')  A   |   |   |   |   |   |   | 1 | 2 | 3 | . |
WRITE(*, '(F10.1)')  A   |   |   |   |   |   | 1 | 2 | 3 | . | 3 |
WRITE(*, '(F10.2)')  A   |   |   |   |   | 1 | 2 | 3 | . | 3 | 5 |
WRITE(*, '(F10.3)')  A   |   |   |   | 1 | 2 | 3 | . | 3 | 4 | 5 |
WRITE(*, '(F10.4)')  A   |   |   | 1 | 2 | 3 | . | 3 | 4 | 5 | 0 |
WRITE(*, '(F10.5)')  A   |   | 1 | 2 | 3 | . | 3 | 4 | 5 | 0 | 0 |
WRITE(*, '(F10.6)')  A   | 1 | 2 | 3 | . | 3 | 4 | 5 | 0 | 0 | 0 |
WRITE(*, '(F10.7)')  A   | * | * | * | * | * | * | * | * | * | * |
WRITE(*, '(F10.4)') -A   |   | - | 1 | 2 | 3 | . | 3 | 4 | 5 | 0 |
WRITE(*, '(F10.5)') -A   | - | 1 | 2 | 3 | . | 3 | 4 | 5 | 0 | 0 |
WRITE(*, '(F10.6)') -A   | * | * | * | * | * | * | * | * | * | * |
                         +---+---+---+---+---+---+---+---+---+---+
```

Real Output – the E Descriptor

Editing Descriptor(s): $r\mathtt{E}w.d$ and $r\mathtt{E}w.d\mathtt{E}e$

- `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`)
- <u>**IMPORTANT**</u>: Before printing, the number to be printed is converted to $\pm 0.dddd \ldots \times 10^n$. The actual printout is $\pm 0.dddd \ldots \times 10^n$ rather than the original.
- For $\mathtt{E}w.d$, the w positions are arranged as follows. Thus, $w \geq d + 7$

```
                  <--- d --->   <---- 3 ---->
+-------+---+---+---------+---+---+---+---+
| ..... | 0 | . | ....... | E |   |   |   |
+-------+---+---+---------+---+---+---+---+
<------------------ w ------------------>
```

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

```
                   <--- d --->   <------ e ----->
+-------+---+---+---------+---+---+------+---+
| ..... | 0 | . | ....... | E |   | .... |   |
+-------+---+---+---------+---+---+------+---+
<-------------------- w -------------------->
```

```
REAL :: PI = 3.1415926

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

```
  1   2   3   4   5   6   7   8   9   10  11  12
+---+---+---+---+---+---+---+---+---+---+---+---+
|   | 0 | . | 3 | 1 | 4 | 1 | 6 | E | + | 0 | 1 |
|   | 0 | . | 3 | 1 | 4 | E | + | 0 | 0 | 0 | 1 |
| 0 | . | 3 | 1 | 4 | 1 | 5 | 9 | 3 | E | + | 1 |
+---+---+---+---+---+---+---+---+---+---+---+---+
```

Logical Output

Editing Descriptor(s): r`L`w

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

```
LOGICAL :: A, B
A = .TRUE.                        1   2   3
B = .FALSE.                     +---+---+---+
WRITE(*,'(L1,L2)')  A, B        | T |   | F |
                                +---+---+---+---+---+---+---+
WRITE(*,'(L3,L4)')  A, B        |   |   | T |   |   |   | F |
                                +---+---+---+---+---+---+---+
                                  1   2   3   4   5   6   7
```

Character Output

Editing Descriptor(s): r`A` and r`A`w

- `A`: `CHARACTER` output
- 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 value is used.
  2. If w is larger than the length of the character value, that value is *right* justified.
  3. If w is smaller than the length of the character value, that value is truncated (*i.e.*, only the *left-most* w characters will be printed).

```
CHARACTER(LEN=5) :: A
CHARACTER(LEN=3) :: B

A = 'ABCDE'                1   2   3   4   5   6   7   8   9   10
B = 'XYZ'                +---+---+---+---+---+---+---+---+
WRITE(*,'(A3,A5)') A, B  | A | B | C |   |   | X | Y | Z |
WRITE(*,'(A3,A)')  A, B  | A | B | C | X | Y | Z |
WRITE(*,'(A,A5)')  A, B  | A | B | C | D | E |   |   | X | Y | Z |
WRITE(*,'(2A)')    A, B  | A | B | C | D | E | X | Y | Z |
                         +---+---+---+---+---+---+---+---+
```

Spacing and Tabbing

Editing Descriptor(s): n`X` and `T`c, `TL`c and `TR`c

- n`X`: insert n spaces
- `T`c: move to position c
- `TL`c: move backward c positions
- `TR`c: move forward c positions
- **<u>IMPORTANT</u>**:
  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
```

Printer Control

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

First Char on a Line	*Effect*
space	Advance to next line **BEFORE** printing
0	Advance two lines **BEFORE** printing
1	Advance to next page **BEFORE** 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

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

The Slash (/) Descriptor

1. Advance to next line.
2. Between two adjacent `/`s, no comma is required. `//` is the same as `/,/`.

```
INTEGER           :: A = 123, B = 456
CHARACTER(LEN=8) :: C = "ABCDEFGH"
CHARACTER(LEN=3) :: ABC
CHARACTER(LEN=7) :: TITLE

                                                              1
                                                     1234567890
WRITE(*, '(1X,I4/I5.4/T3,A)') A, B, C                  123
                                                      0456
                                                       ABCDEFGH
TITLE = ' TITLE '
ABC   = 'ABC'
WRITE(*,'(//1X, A, A/)') TITLE, ABC              123456789012
                                                    |
                                                    |
                                                       TITLE ABC
                                                    |
             next WRITE prints here -->             |
```

The Grouping Descriptor: $r(\ldots)$

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
```

Example 2

Print a multiplication table.

```
PROGRAM  Multiplication_Table
   IMPLICIT  NONE
   INTEGER, PARAMETER :: MAX = 9
   INTEGER            :: i, j
   CHARACTER(LEN=80)  :: FORMAT

   FORMAT = "(9(2X, I1, A, I1, A, I2))"
   DO i = 1, MAX
      WRITE(*,FORMAT) (i, '*', j, '=', i*j, j = 1, MAX)
   END DO
END PROGRAM  Multiplication_Table
--------------------------------------------------
```

```
  1*1= 1  1*2= 2  1*3= 3  1*4= 4  1*5= 5  1*6= 6  1*7= 7  1*8= 8  1*9= 9
  2*1= 2  2*2= 4  2*3= 6  2*4= 8  2*5=10  2*6=12  2*7=14  2*8=16  2*9=18
  3*1= 3  3*2= 6  3*3= 9  3*4=12  3*5=15  3*6=18  3*7=21  3*8=24  3*9=27
  4*1= 4  4*2= 8  4*3=12  4*4=16  4*5=20  4*6=24  4*7=28  4*8=32  4*9=36
  5*1= 5  5*2=10  5*3=15  5*4=20  5*5=25  5*6=30  5*7=35  5*8=40  5*9=45
  6*1= 6  6*2=12  6*3=18  6*4=24  6*5=30  6*6=36  6*7=42  6*8=48  6*9=54
  7*1= 7  7*2=14  7*3=21  7*4=28  7*5=35  7*6=42  7*7=49  7*8=56  7*9=63
  8*1= 8  8*2=16  8*3=24  8*4=32  8*5=40  8*6=48  8*7=56  8*8=64  8*9=72
  9*1= 9  9*2=18  9*3=27  9*4=36  9*5=45  9*6=54  9*7=63  9*8=72  9*9=81
```

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:

```
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): r`I`w

- `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 <u>**IGNORED**</u> 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): r`F`$w.d$

- `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 right-most 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 `F`$w.d$ has no effect.

```
REAL :: A, B, C                          1    1    2
                                12345678901234567890 1
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.

3. 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): r`A` and r`A`w

- `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): n`X` and `T`c. Note that no repetition can be used.

- n`X`: insert n spaces
- `T`c: 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 <u>**IGNORED**</u>.
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
```

Example 5

Read in two arrays and compute their inner product.

```
PROGRAM  InnerProduct
   IMPLICIT  NONE
   INTEGER, PARAMETER          :: SIZE = 50
   INTEGER                     :: Number, Sum, i
   INTEGER, DIMENSION(1:SIZE) :: x, y
   CHARACTER(LEN=80)           :: Fmt1, Fmt2, Fmt3

   Fmt1 = "(I5/(2I5))"
   Fmt2 = "(1X,A//1X,3A5/(1X,3I5))"
   Fmt3 = "(/1X, A, I7)"

   READ(*,Fmt1)   Number, (x(i), y(i), i = 1, Number)
   WRITE(*,Fmt2)  "Input Data", "No", "X", "Y", &
                  (i, x(i), y(i), i = 1, Number)
   Sum = 0
   DO i = 1, Number
      Sum = Sum + x(i)*y(i)
   END DO
   WRITE(*,Fmt3)  "Input product = ", Sum
END PROGRAM  InnerProduct
------------------------------------------------------------
Input:    1234567890    Output:
          5                  Input Data
          4    7
          2    -1               No    X    Y
          6    2                 1    4    7
          -3   4                 2    2   -1
          -2   -3                3    6    2
                                 4   -3    4
                                 5   -2   -3

                              Input product =      32
```