Eg 6.1

```
REAL    A, B ; INTEGER  I
CHARACTER (len=7) :: Ch ; COMPLEX Z
! CHARACTER*7  Ch
! CHARACTER   Ch*7
OPEN (1,   FILE = 'input.dat')
OPEN (2, FILE = 'output.dat')
READ(1, *)  A, I, B, Ch, Z
```

Aside:

A = 3.1

I = 2

B = 1.ϕEϕ4 = 1ϕϕϕϕ

Ch = 'nameϐϐϐ'

Z = (2.1, −3.2) = 2.1 − i 3.2

($i = \sqrt{-1}$)

If instead of name we have

naϐme ← "blank"

then use of * for formatting would fail: Ch = 'naϐϐϐϐϐ' but the READ would enter "me" for Z.

WRITE(2,*) A, I, B, Ch, Z

For Z with * formatting we get

(2.1, -3.2)

WRITE(2,10) A, I, B, Ch, Z ! alternative

10 FORMAT(F5.1, I3, E8.1, A, 2F5.1)

bb3.1	bb2	bb.1E+05	nameb6b	bb2.1	b-3.2
A	I	B	Ch	ReZ	ImZ

not an ordered pair

We must supply (,) in the case of ordered pair notation for Z:

10 FORMAT(..., A, '(', F5.1, ',', F5.1, ')')

This gives: ... | nameb6b | (bb2.1, b-3.2)
Ch | Z

Could use F0.1, F0.1.

Eg 6.2 (1)

```
OPEN (1, FILE = 'myfile.dat')
DO J=1,1000  ! limit # of iterations to 1000.
READ (1,*) K,L,M,N
IF (K==99) exit  ! exit from Do-loop.
< block of statements to process K,L,M,N >
ENDDO
```

Eg 6.2 (2)

```
OPEN (1, FILE = 'myfile.dat')
DO J= 1, 1000
READ (1, *, END=2)  K, L, M, N
< process K,L,M,N >
ENDDO
2 CONTINUE
```

Eg 6.3 (1)

```
REAL  A(24)
! REAL, DIMENSION(24) :: A
! DIMENSION A(24) ! using IMPLICIT typing
OPEN(1, FILE = 'myfile.dat')
READ(1,*) A  ! List-directed/free format
```

Eg 6.3(2)

Again, simply

```
READ(1,*) A
```

Aside: If the elements in the file are arranged by column in the file there is a problem (unlikely):

A(1)	A(7)	A(13)	A(19)
A(2)	A(8)	A(14)	A(20)
⋮	⋮	⋮	⋮
A(6)	A(12)	A(18)	A(24)

} values in data file.

Eg 6.3(3)

```
REAL, DIMENSION (100) :: A = 999.
! A has elements A(1), ..., A(100)
! Each element is initialised to 999.

OPEN(1, FILE='myfile.dat')

READ(1,*,END=1) A

! DO I=1,100
  IF (A(I) == 999.) EXIT

  ENDDO
  Nitems = I-1
```

Alternatives: if data is regularly spaced

Case of 1 value each row:

```
    DO I=1,100
    READ (1,*,END=1) A(I)
  ENDDO
1 Nitems = I-1.
  ! No need to initialise A.
```

Case of 2 values each row:

```
      DO  I = 1,100,2
      READ (1,*, END=1)  A(I), A(I+1)
      ENDDO
1     Nitems = 2*(I-1)
```

Warning about roundoff in tests:

in the original example the test used was:

```
IF (A(I) == 999.) EXIT
```

Generally care is needed comparing REAL numbers: a more robust test is

```
IF( ABS(A(I)- 999.) .LT. 0.5E-3) EXIT
```

↑ small tolerance

Eg 6.4 2D Arrays

```
1 2
3 4   ⟶   [1 2]
5 6        [3 4]
           [5 6]
```

```
INTEGER A(3,2)
! INTEGER, DIMENSION(3,2) :: A
OPEN(1,FILE='myfile.dat')
READ(1,*) A   ! WRONG!
```

We get for A: $\begin{bmatrix} 1 & 4 \\ 2 & 5 \\ 3 & 6 \end{bmatrix}$

The items are read from the file in row order, i.e. 1 2 3 4 5 6.
However, the values are assigned to the elements of A in column order:
A(1,1), A(2,1), A(3,1), A(1,2), A(2,2), A(3,2)
– FORTRAN is column oriented.

Either the data in the file must be re-ordered or different FORTRAN coding is required.

§6.4 (more) 2D Arrays

```
READ(1,*) ((A(I,J),J=1,2),I=1,3)
```

OR

```
DO I=1,3
  READ(1,*) (A(I,J),J=1,2)
ENDDO
```

OR

```
DO I=1,3
  READ(1,*) A(I,:)
ENDDO
```

NEW CODE

```
INTEGER  A(3,2)
OPEN(1, FILE='myfile.dat')
READ(1,*)  ((A(I,J), J=1,2), I=1,3)
```

To trap the EOF if necessary:

```
DO  I=1,3   ! to READ rows of file
READ (1, *, END=10)  (A(I,J), J=1,2)
ENDDO
   :
10 CONTINUE   ! or some other statement
```

Use TRANSPOSE function:

```
INTEGER  A(3,2), B(2,3)
OPEN(1, ...
READ (1,*) B
A = transpose(B)
```

Drawback is the use of 2 arrays.

$A = B^T$ ← 'transpose'

Eg 6.5

(1) Discard header in file:

```
INTEGER A(3,2)
OPEN(1, FILE='my file.dat')
READ(1,*)  ! strips line
READ(1,*) ((A(I,J), J=1,2), I=1,3)
```

(2) Store header for re-use:

```
CHARACTER(80) :: HEADER
! Assume ≤80 characters in HEADER
OPEN(1,...
READ(1,'(A)') HEADER
READ(1,*) ((A(I,J), J=1,2), I=1,3)
```

So HEADER = 'Any␣heading␣line....'
(80 characters)

```
READ(1,10) HEADER
10 FORMAT(A)
```

(3) Embedded headers/labels:

In file we have

A␣=␣34.2␣B␣=␣21.72␣␣C␣=␣(13.3,-4.2)

```
REAL A,B; COMPLEX C; CHARACTER Ch*3
READ(1,*) Ch, Ch, A, Ch, Ch, B, Ch, C
```

'A␣␣' '=␣␣' 34.2 'B␣␣' '=␣␣' 21.72 'C=␣' complex number as ordered pair

Here the ␣␣ labels A,B,C are discarded.

Note this is <u>not</u> "A␣=" ! Due to * format.

(4) Embedded headers/labels which are kept. Use the same line of the data file as in (3).

```
REAL A,B ; COMPLEX C
CHARACTER    Any(5)*3
```

size of array — length of each element

```
! Any is a CHARACTER array
! of 5 elements Any(1),...,Any(5).
! Each element is a CHARACTER
! variable of length 3.

READ(1,*) Any(1:2), A, Any(3:4), B, Any(5), C
! Any(1:2) is a section of the array
! Any with elements Any(1), Any(2).
```

Any(1) = 'Abb', Any(2) = '=bb'
Any(3) = 'Bbb', Any(4) = '=bb'
Any(5) = 'C=b'

To output the values of A, B, C with labels evenly spaced:

```
WRITE(*,*) Any(1:2), A, Any(3:4), B, &
           Any(5)(1:1), 'bb=bb', C
```

Output is:

```
A bb= bb 34.2 Bbb=bb 21.72 Cbb=bb ( , )
```

depends exactly on compiler.