1 **FORTRAN 77**
Chapter 2
Satish Chandra

2 **Input and Output Statement**
- I/O statement consists of two parts; an executable part and a declaration part.
- The executable part commands certain items to be read into memory or read out of it.
- The declaration part gives information on the exact form in which data will be found in the input or to be produce at output.

3 **Input and Output Statement**
- To make assignments and do computations, we need input for the calculations and to output the results.
- For inputting and outputting data in Fortran language, we use READ and WRITE statements

4 **Input and Output Statement**
- Read is used for input, while write is used for output. A simple form is
  read (unit no, format no) *list-of-variables*
  write(unit no, format no) *list-of-variables*
- The unit number can refer to either standard input, standard output, or a file. The format number refers to a label for a format statement.
- When format is not specified the input-output statement is called list-directed or "format-free".

5 **List-Directed Input**
- The term list-directed input means that *the types of the variables in the variable list determine the required format of the input data*.
- The READ statement is in the form:
  read (*, *) *list-of-variables*
  where the list of variables is the list into which the values being read are stored.
- The (*, *) means the input data can be read from the standard input, in a flexible form.

6 **List-Directed Input**
- For example:
```
read(*,*) x, y, z
read(*,*) height, weight
read(*,*) 'radius=', r
```
- During the execution the data are typed in the same line separated by comma or space or line by line.
```
read (*,*) i, j
read (*,*) k,l
```

7 **List-Directed Output**
- The term list-directed implies that *the types of the values in the output list of the write statement determine the format of the output data*.
- The WRITE statement is in the form:
  write (*, *) *list-of-variables*
  where the list of variables is the list into which the values being write are stored.
- The (*, *) means the output data can be printed to the standard output, in a flexible form

8 **List-Directed Output**
- For example:
```
write(*,*) x, y, z
write(*,*) height weight
write(*,*) 'area of triangle=', a
```
- During the execution the data are typed in the same line separated by comma or space or line by line.
```
write (*,*) i, j
write (*,*) k,l
```

9 **Example1:**

```
C     PROGRAM 1
      real cent, fahr
C           This is a program to convert temperature   from centigrade to fahrenheit
      read(*,*) 'Temperature in centigrade =', cent
      fahr=1.8 * cent +32.0
      write(*,*) 'Temperature in fahrenheit =', fahr
      stop
      end
```

10 **Formatted Statements**

- So far we have only showed free format input/output. This uses a set of default rules for how to output values of different types (integers, reals, characters, etc.). Often the programmer wants to specify some particular input or output format, e.g. how many decimals in real numbers. For this purpose Fortran 77 has the format statement. The same format statements are used for both input and output.

11 **Formatted Output statements**

- The format statement is a non-executable statement. It merely supplies information to the compiler on how data will be found and how results are to be presented.
- The syntax is
  write(*, label) list-of-variables
  label format (format-specifications)
- For example: write(*,10) a,b,c
  10 format(format specifications)

12 **Format Specifications**

- X – Specification
- I – Specification
- F – Specification
- E – Specification
- A – Specification
- H – Specification
- T – Specification

13 **X – Specification**

- X code directs the computer to skip the specified number of spaces in the output. The general form is nX. Here n is the number of positions to be skipped.
- For example:

```
    write(*,10)
10  format(1x,'Good',3x,'morning',1x,'to',2x,'You')
```

- The display will be –
  _Good___morning_to __You

14 **I – Specification**

- This is for displaying integer type data . The general form is Iw, here w denotes field width, which includes the sign of the number. Eg 1.

```
      write(*,20) m, n
20    format(i5, i6)
```

15 **I – Specification**

- Eg 1.

```
      write(*,20) m, n
20    format(i5, i6)
```

- If m=-123 and n=2345 then the display will be;
- -123 2345
- Eg 2.

```
      write(*,30) m, n
```

30 format(i4, 5x, i4)

- -123 2345

16 **F – Specification**

- It is used for writing real data in decimal form. The specification is Fw.d. Here w is the filed width (the number of spaces including the decimal point and sign) and d is the number of decimal places. Eg.

```
write(*,15) x, y
15 format(f8.2, f10.3)
```

17 **F – Specification**

- Eg 1.

```
write(*,15) x, y
15 format(f8.2, f10.3)
```

- If x=12.346, y=-2345.6 then the display will be;
- 12.35 -2345.600
- Eg 1.

```
write(*,20) x, y
20 format(f5.2, 3x, f9.3)
```

- 12.35 -2345.600

18 **F – Specification**

number input	format specification	display
13.45	f5.0	13.0
13.45	f5.1	13.5
13.45	f5.2	13.45
13.45	f4.2	(field width error)
13.45	f6.3	13.450
-13.45	f5.2	(field width error)

19 **I & F – Specifications**

- If the variables are of the same nature and specification repetition can be avoided as follows

```
write (*,10) m, n, o
10 format (3i5)
```

- Here instead of i5 being repeated thrice 3i5 is used. If m = -123, n = 456 and o = 789 the display will be; -123 456 789. Similarly

```
write (*,12) m, n, o
12 format (1x, 3(i3,2x))
```

20 **I & F – Specifications**

- Similarly, if x =-23.2, y =1.23, z =43, k = 23.3

```
write (*,10) x, y, z, k
10 format (1x,3f6.2,i5)
```

- Here f6.2 is repeated thrice and i5 once. The display will be; -23.20 1.23 43.00 23
- But with

```
write (*,20) x, y, z, k
20 format (1x,3(f6.2,2x),i5)
```

- -23.20 1.23 43.00 23

21 **E – Specification**

- This is used for real data in the exponent form. The syntax is Ew.d, where w is the field width which should be minimum 7. The usage is as that of F, i.e. the filed width w is the number of spaces including the sign and decimal point of mantissa, e, sign and power of exponent) and d is the number of decimal places in mantissa. Eg.

```
write(*,15) x, y
15 format(e9.2, e10.3)
```

22 **E – Specification**

- Eg. 1

```
    write(*,15) x, y
15 format(e9.2, 2x, e10.4)
```

- If $x=1.34 \times 10^{-3}$, $y=-3.3456 \times 10^{-12}$ then the display will be; 0.13e-02 -0.346e-11
- Eg. 2

```
    write (*,10) x, y, z,
10 format (1x, 3(e8.2, 1x))
```

23 **A – Specification**

- It is used to write character type data. The specification is in the form Aw . Suppose name is a character type variable of width 20 (It is declared as Character *20 name) then the value (the string of characters) stored in name is displayed as

```
    write (*, 13) name
13 format (1x, a20)
```

- If, name="PPN, KANPUR" then the display will be; PPN, KANPUR

24 **A – Specification**

- The width can be greater than the actual size. For in the above case format(1x, a25) can be used. But if the width assigned is smaller the name will be truncated. For example, if use the following format

```
    write (*, 15) name
15 format(1x, a8)
```

- then the display will be; PPN, KAN

25 **H – Specification**

- The general form is nH and it is for writing a string of characters instead of the single quote.

```
    write(*,20)
20 format (1x, 18hPHYSICS DEPARTMENT)
```

- will display PHYSICS DEPARTMENT
- If 16H is used then the last two characters N and T will not be displayed.

26 **T – Specification**

- The syntax is Tn. Here the display is at the nth position (i.e. n-1 spaces are skipped)

```
    write (*, 19) x
19 format (1x, t20, f6.3)
```

- Here the value of x is written at the 20th column of the screen.

27 **The forward slash / specification**

- It signals the end of output and the control is transferred to the next line.

```
    write (*, 16) x, y
16 format (1x, f8.3/ f8.3)
```

- causes the display of x and y in two lines.
- If we use / twice then they are displayed alternatively

28 **The Backward slash \ specification**

- This is placed at the end of a FORMAT statement and it causes the next input or output to read and write in the same line. For example

```
    write (*, 17)
16 format (1x, 'Input the number:' \ )
    read (*, *) n
```

- Here after the message ' Input the number' the cursor flickers at the end of the line, waiting for the input.

29 **Formatted Input Statement**

- All the format for writing data can be used for reading too. But practically formatted reading is a tiresome process. So we use format free statements for inputting data.

30 **ENDS**

CHAPTER 2