SCSJ 2733
Fortran – Data Files

Mohsin Mohd Sies

Faculty of Chemical and Energy Engineering

Outline

- Motivation
- OPEN file
- Communicate with file
- CLOSE file

Motivation

Running programs fully on the terminal console (computer screen) can be very limiting

- *Permanent record* – Usually we want program output to be saved for
  - Later reference
  - Further analysis
- *Large scale* – Usually input data is too huge to be typed by hand for each run
- We need to use data files for these purposes.

Computer Screen – Default I/O Device

- Also known as *console, terminal, monitor*
- `PRINT *` or `READ *` refer to the default I/O device

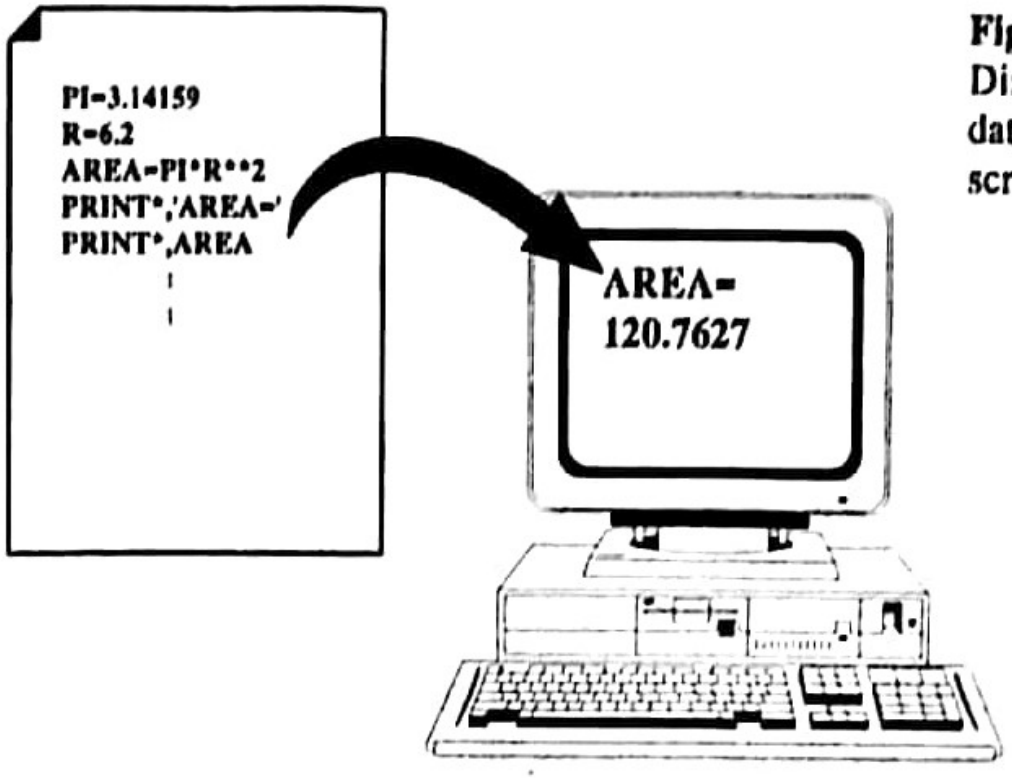


Fig. 9-1
Direct write of data to the CRT screen

Writing to a data file

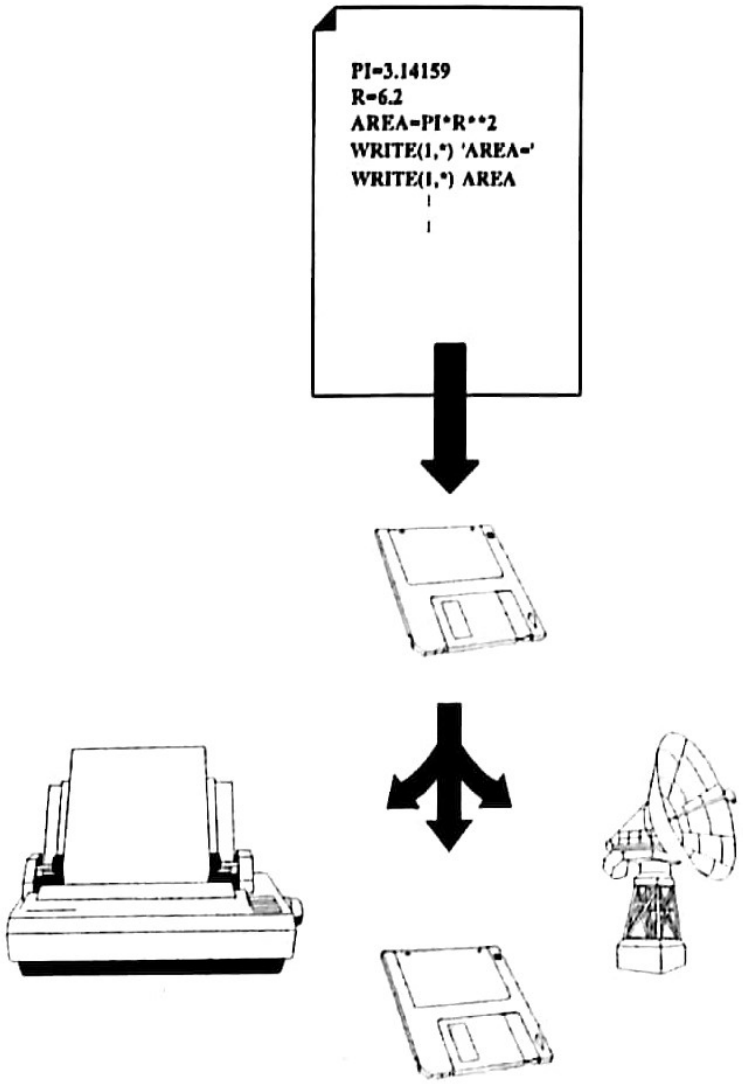


Fig. 9-2
Writing to a data file.

Writing to a data file

EXAMPLE 9.1

Here is a simple example of how to write to a data file named EXPER.DAT:

```
OPEN(UNIT=8, FILE='EXPER.DAT', STATUS='NEW')
WRITE(8,*) DIST, TIME, VELOC
CLOSE(8)
```

Reading from a data file

EXAMPLE 9.2

In the following example, we will read data from a file named 'NOBEL.DAT' and assign the data to the variables WEIGHT, MASS, and DENSIT. Notice that the file must already exist in order to read from it.

```
OPEN(UNIT=3, FILE='NOBEL.DAT', STATUS='OLD')
READ(3,*) WEIGHT, MASS, DENSIT
CLOSE(3)
```

Handling a data file

- Three step process
  - OPEN the file
  - Manipulate the file
  - CLOSE the file when finished

- OPEN statement mainly contains
  - The file name
  - Assigned reference number
  - Status of file

```
OPEN(UNIT=3, FILE='NOBEL.DAT', STATUS='OLD')
READ(3,*) WEIGHT, MASS, DENSIT
CLOSE(3)
```

The READ or WRITE statement

```
OPEN(UNIT=3, FILE='NOBEL.DAT', STATUS='OLD')
READ(3,*) WEIGHT, MASS, DENSIT
CLOSE(3)
```

- `READ (3,*)` means
  - *read* from unit 3 which is actually the file NOBEL.DAT and
  - * means the data is free format. (the three data are separated by space or comma)

- The `WRITE` statement works similarly (* means free format)
- `READ` is usually not formatted, but we usually want to `WRITE` in a specified format
- `READ *,` is equivalent to `READ (*,*)`
- `PRINT *,` is equivalent to `WRITE (*,*)`

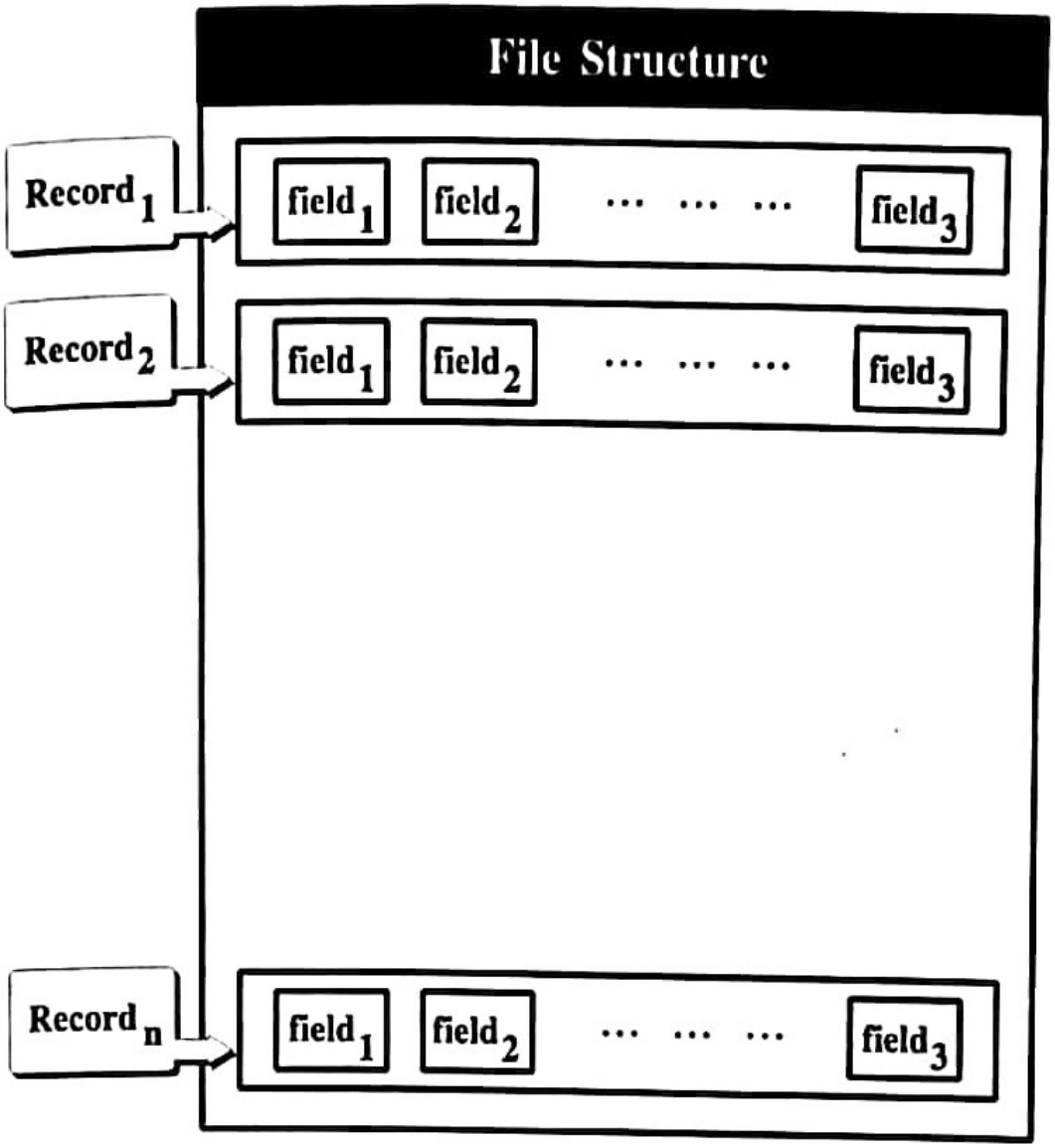
The formatted WRITE statement

```
      OPEN (UNIT=3, FILE='OUTPUT.TXT', STATUS='NEW')
      WRITE (3,10) WEIGHT, MASS, DENSIT
      CLOSE (3)
10    FORMAT('', F7.2,F9.6,F10.1)
```

- `WRITE (3,10)` means
  - *write* to unit 3 which is actually the file OUTPUT.TXT and
  - `10` means refer to statement label `10` for the formatting specifications of the output

- Statement line `10` can be written anywhere in the program (even after `CLOSE`)
- The format line follows the rules as discussed in the earlier chapter on input/output.

Data file Terminologies – Records and Fields



Data file Terminologies – example file

A12KM0110	80.0	92.2	93	32	55.71	86
A12KM0112	78.5	88.9	84	40	49.05	98
A12KM0113	72.3	88.9	85	46	54.76	86
A12KM0114	73.8	87.8	95	44	73.33	92
A12KM0115	80.0	88.9	94	38	54.29	94
A12KM0117	80.0	91.1	93	64	68.57	98
A12KM0119	78.5	91.1	86	36	52.38	98
A12KM0120	83.1	91.1	88	64	81.90	94

Data file types

- SEQUENTIAL
  - Reads records in the *order* that they are written in the file
  - Writes new data at the end of the file
- DIRECT ACCESS
  - Can read from anywhere in the file
  - Can write to anywhere in the file (overwrites existing data) without disturbing any other record
- Engineers and scientists usually deal with sequential files

Sequential file OPEN options

```
OPEN ([UNIT=] integer expression, [FILE=] string,
      [ACCESS=string,]
      [ACTION=string,]
      [BLANK=string,]
      [DELIM=string,]
      [ERR=statement label n,]
      [FORM=string,]
      [IOSTAT=integer variable,]
      [PAD=string,]
      [POSITION=string,]
      [RECL=integer expression,]
      [STATUS=string,])
```

- Only a few options are usually needed and used

Sequential file OPEN options

- File NAME as a variable

EXAMPLE 9.5

You may specify the file name to be a variable that is read in at execution time, provided that you have properly set up the name as a character variable. In the following example, we use the character string stored in NAME to create the desired file entered by the user:

```
CHARACTER*10 NAME
PRINT *, 'Enter output file name:'
READ *, NAME
      ⋮
OPEN(UNIT=16, FILE=NAME,...)
WRITE (16,*) X, Y
      ⋮
```

Sequential file OPEN options - STATUS

- STATUS='NEW'
  - Creates new file for WRITE
- STATUS='OLD'
  - Use an existing file
  - READ requires this
  - WRITE can also use this
- STATUS='SCRATCH'
  - Creates a temporary file
- STATUS='UNKNOWN'
  - If the file exists, it will be treated as OLD
  - If the file doesn't exist, it will be treated as NEW

- If STATUS is omitted, it will be assumed as UNKNOWN

Closing the sequential file

- Files have to be closed after use
- The `CLOSE` statement can be anywhere in the program after it has been used but before the `END` statement of the program
- If the file is assigned to `UNIT 3` (for example), then
  - `CLOSE (UNIT=3)` is equivalent to `CLOSE (3)`

Example

9.6 Write a program to read a file CONCENTR.DAT whose records contain concentrations of various species, [*a*], [*b*], and [*c*] in a chemical reaction. Assume that the data are written to the file with the format specifier (3F10.6). Read in one record at a time, and for each set of concentrations compute the rate constant defined by

$$k = \frac{[a][b]}{[c]}$$

Print the results ([*a*], [*b*], [*c*], and *k*) to the screen. If an error occurs during the read operation, print the message "Input Error!" When the end-of-file occurs, terminate the program and print the message "Calculations Complete."

Example

```
C*************************************************************
C Open the file and then read one set of [a], [b], and [c]
C values at a time. Use these to compute k and print the
C results. Include in the READ statement the options ERR=
C and END=. These will detect read errors and end-of-file
C respectively.
C*************************************************************
      REAL K
      OPEN(UNIT=1, FILE='CONCENTR.DAT', STATUS='OLD')
      DO WHILE( .TRUE.)
         READ(1,10, ERR=20, END=30) A, B, C
10       FORMAT(3F10.6)
         K=A*B/C
         PRINT *, A, B, C, K
      END DO
20    STOP 'Input Error!'
30    STOP 'Calculations Complete'
      END
```

Example

'GROWTH.DAT'

```
10/12/93        23:47   1234568
10/13/93        11:47   1458450
10/14/93        06:34   1534389
10/15/93        09:23   1637238
<end of file>
```

'POTENCY.DAT'

```
10/12/93        23:47   147.345
10/13/93        11:47   146.234
10/14/93        06:34   148.346
10/15/93        09:23   147.225
<end of file>
```

The first two columns in each file represents the date and time, respectively, at which the data were taken. The third column represents the number of cells (in 'GROWTH.DAT') and their potency (in 'POTENCY.DAT'). Write a program that opens both files and merges them into a new file 'BACTERIA.DAT' with the following format:

```
10/12/93        23:47   1234568         147.345
10/13/93        11:47   1458450         146.234
10/14/93        06:34   1534389         148.346
10/15/93        09:23   1637238         147.225
<end of file>
```

Example

```
C*************************************************************
C Open all three files at the same time, but give each a
C different unit number.
C*************************************************************
      CHARACTER DATE*12, TIME*7
      REAL NUMCEL
      OPEN(UNIT=1, FILE='GROWTH.DAT', STATUS='OLD')
      OPEN(UNIT=2, FILE='POTENCY.DAT', STATUS='OLD')
      OPEN(UNIT=3, FILE='BACTERIA.DAT', STATUS='NEW')
C*************************************************************
C Set up a loop to read the data from GROWTH.DAT and POTENCY.DAT
C and merge them. Be sure to remove redundant information before
C writing it to the new file. If an end-of-file specification
C is encountered, stop.
C*************************************************************
      DO WHILE (.TRUE.)
         READ(1,*, END=10) DATE, TIME, NUMCEL
         READ(2,*, END=10) DATE, TIME, POTENC
         WRITE(3,*) DATE, TIME, NUMCEL, POTENC
      END DO
10    STOP 'End of Input Data'
      END
```