

1. Elements of Fortran I/O

Fortran has a rich set of I/O keywords, which include:

- **PRINT**: Restricted version of WRITE that works only with the screen (rather than any file or device)
- **formatting**: various format specifiers that we will cover in turn
- **WRITE**: put output to file or device
- **READ**: get input from file or device
- **FORMAT**: specify a format for I/O
- **OPEN**: open a file or device for read/write
- **CLOSE**: close a file or device (done I/O to device)

2. The PRINT statement

Format: PRINT *fmt* [,list]

- [,list]: list of variables and constants/literals to be printed
- *fmt*: Either a format specifier, or use default
 - *: use system default format for the provided variable or literal
 - PRINT *, 'NUMBER = ', num, ', answer = ', ans
 - Format specifier (next several slides) such as:
 - PRINT '("NUMBER = ", i3, ", answer = ", F10.2)', num, ans

The above lines will print (respectively, assume num=2, ans=3.78):

```
NUMBER =          2 , answer =    3.780000
NUMBER =  2, answer =      3.78
```

3. Format Specifiers by type

A *format* specifier is used when you wish to force the I/O to have a certain pattern (i.e. print into a space 10 characters wide, use only 2 decimal points, etc). Format specifiers can be given as FORMAT statements, or as string literals in place of second * in READ/WRITE or the only * in PRINT.

Each specifier has a leading character that determines the type:

- **I**: an Integer variable/constant
- **F**: a real (Floating point) variable/constant
- **E**: scientific notation (Exponentiation)
- **ES**: scientific notation wt first digit 1
- **A**: a character (Alphanumeric) variable/literal

Examples:

- PRINT '(i3,i5)', j, k
- WRITE (*, '(a10)'), name
- 100 FORMAT (2F9.2)

4. Formatting Integers

Format: [*r*]*Iw* [*m*]

r: repeat count; *w*: number is right justified in *w* column space;

m: number has leading zeroes added until it has at least *m* digits

- Can insert spaces between specifiers wt *rX* (3X inserts 3 spaces)

```
!                               0      1      2
! Column count comment:         123456789012456789012345
PRINT '(i4,i6)', 2, 12345  ==>  '  2 12345'
PRINT '(2i8.3)', 2, 12345  ==>  '    002  12345'
PRINT '(2i4.4)', 2, 12345  ==>  '0002****'
PRINT '(i3.3,2X,i4.4)', 3, 8 ==>  '003  0008'
PRINT *, 2, 12345          ==>  '                2          12345'
! Column count comment:         123456789012456789012345
```

5. Formatting decimal numbers (reals and doubles)

Format: $[r]Fw.d$

r : repeat count;

w : number is right justified in w column spaces, w must include space for decimal and at least 1 leading digit, and sign if negative;

d : number of decimal places to print (round to): $w \leq d + 2$, if positive;

$w \leq d + 3$ if negative !

- Can insert spaces between specifiers wt rX (3X inserts 3 spaces)

```
!
! Column count comment:           0      1      2
PRINT '(":",F7.2,2x,F6.4)', 3.679, 0.12345 ==> : 3.68 0.1235
PRINT '(F5.2,2x,F5.2)', 12.3456, 1.23456 ==> 12.35 1.23
PRINT '(F5.2,2x,F5.2)', -12.3456, -1.23456 ==> ***** -1.23
PRINT '(F4.2,2x,F4.2)', 12.3456, 1.23456 ==> ***** 1.23
PRINT '(F4.2,2x,F4.2)', 123456D-4, 123456D-5 ==> ***** 1.23
PRINT '(2F10.3)', 12.34567, 1.234567 ==>          12.346      1.235
! Column count comment:           123456789012345678901234
```

7. Formatting strings and characters

Format: $[r]A[w]$

r : repeat count;

w : number is right justified in w column spaces. If string length is greater than w , then only the first w characters are printed.

- Can put string literals inside format descriptor

```
!
! Column count comment:           0      1      2
PRINT '"HELLO", 3X, A, "!"', 'Richard' ==> HELLO Richard!
PRINT '"Hello ", A4, A1)', 'Richard', '!' ==> Hello Rich!
PRINT '(3A8)', 'Hello', 'Richard', '!' ==>      Hello Richard      !
! Column count comment:           123456789012345678901234
```

6. Formatting scientific notation (reals and doubles)

Format: $[r]Ew.d[Ee]$

r : repeat count;

w : number is right justified in w column spaces, w must include space for decimal and at least 1 leading digit, and sign if negative;

d : number of decimal places to print (round to)

e : number digits in exponent

```
!
! Column count comment:           0      1      2
PRINT '(2E10.3)', 123456789.01, 1.2345678901 ==> 0.123E+09 0.123E+01
PRINT '(2E10.3)', -123456789.01, -1.2345678901 ==> -0.123E+09-0.123E+01
PRINT '(2E12.3E3)', 123456789.01, 1.2345678901 ==> 0.123E+009 0.123E+001
PRINT '(E20.15)', 123456789.01 ==> .123456792000000E+09
PRINT '(E20.15)', 123456789.01d0 ==> .123456789010000E+09
! Column count comment:           123456789012345678901234
```

Format $[r]ESw.d[Ee]$ gets rid of leading 0:

```
! Column count comment:           123456789012345678901234
PRINT '(2ES10.3)', 123456789.01, 1.2345678901 ==> 1.235E+08 1.235E+00
PRINT '(2E10.3)', 123456789.01, 1.2345678901 ==> 0.123E+09 0.123E+01
```

8. The FORMAT Statement

For format specifiers that are reused or complicated, can use a **format statement** (usually grouped together after all executable statements).

<label> FORMAT(fmt list)

```
PROGRAM colprint
                                LAST NAME FIRST NAME NET WORTH
                                =====
PRINT 1000                      Bridgefor   Robert $ 10800.50
PRINT 2000                      Kahan      Christophe $ 5000.27
PRINT 3000, 'Bridgeford', 'Robert', &      Schmidt    Robert $    0.75
                                10800.50
PRINT 3000, 'Kahan', 'Christopher', &
                                5000.27
PRINT 3000, 'Schmidt', 'Robert', &
                                0.75

1000 FORMAT ('LAST NAME FIRST NAME NET WORTH')
2000 FORMAT ('=====')
3000 FORMAT (a9, 2x, a10, ' $', F9.2)
```

END PROGRAM colprint

9. Opening files for I/O with the OPEN statement

OPEN ([unit=]# [,optional args])

- **unit=#:** unit # to connect to opened file or device
 - unit=0 is *usually* standard error (screen)
 - unit=5 is *usually* standard in (key-board)
 - unit=6 is *usually* standard out (screen)
 - We will use * instead of using 5 or 6
 - We will use other #'s when opening our own files
- **iostat=ios:** ios is int var, set to 0 if open works, else positive number.
- **err=lab:** statement label to jump to on error opening file
- **file=fnam:** fnam is character expr with file name to open
- **status=st st** is one of:
 - 'old': file already exists
 - 'new': file does not already exist, will be created
 - 'replace': deletes file if it exists, creates new
 - 'scratch': temp file goes away at end of program
 - 'unknown': default; file treated in system-dependent way
- **action=act, act** is one of:
 - 'read': file is read only
 - 'write': file is written only
 - 'readwrite': file is both read and written

```
OPEN(unit=8, file='input_file.dat', status='old', action='read')
READ(8, *), I, J, K
```

11. Simple READ

The simplest version of READ takes only a format specifier (fmt, which can be set to the compiler default by specifying *, and a list of variables to read. This version is of form:

```
READ fmt [, <var list>]
```

```
READ (unit#, fmt) [, <var list>]
```

- **fmt** is either the label of a FORMAT statement, or a string literal containing the format specifier
 - If provided as *, use standard format based on type
 - READ(*, 1000), I1, I2; READ(*, '(2I4)') I1, I2
- **unit#** is either a unit number (see OPEN) or * for keyboard
- The following forms both mean read using standard format from keyboard:
 - READ(*,*) [, <list>]
 - READ * [, <list>]

10. Additional optional parameters to OPEN

OPEN ([unit=]# [,optional args])

- **access=acc:** acc is one of 'sequential' or 'direct' (default 'sequential')
- **form=fm:** fm is one of:
 - 'formatted': file is human readable
 - 'unformatted' : file is opt for system access (not human readable)
- **position=pos:** pos is one of:
 - 'rewind': start at beginning of file
 - 'append': start at end of file
 - 'asis': if file already open, keep in same place, otherwise undetermined
- **recl=r1:** r1 is a pos int giving record length. Mainly used for direct-access files.
- **blank=b1:** b1 is one of (formatted I/O only)
 - 'null' : ignore blanks in numbers
 - 'zero' : treat blanks as zeros
- **delim=del:**
 - 'quote': strings contained in double quotes (")
 - 'apostrophe' : strings contained in single quotes (')
 - : if writing, internal single/double quotes will be doubled
 - 'none': use no delimiters (string ended by EOL?)
- **pad=pd,** pd is one of (def 'yes'):
 - 'yes' : pad input records wt blanks to make fit storage and input format
 - 'no': input record must not be less than associated format

12. Unsimple READ

```
READ([unit=]unit#, [fmt=]fmt [,iostat=ios] [,err=err-lab] &
      [,end=end-label]) [var list]
```

- **iostat:** (*iostatus*) ios is an integer variable name, which will be set to zero on success, nonzero on error, negative if end-of-record or end-of-file.
- **err=lab:** jump to label lab on error that causes a positive ios
- **end=lab:** jump to label lab on error that causes a negative ios

```
INTEGER :: ierr, I, J, K
```

```
READ (unit=8, fmt=1000, iostat=ierr) I, J, K
```

```
! READ (8, 1000, iostat=ierr) I, J, K           ! same as line above!
```

```
IF (ierr .NE. 0) THEN
```

```
    PRINT *, 'READ statement returned error: ', ierr
```

```
    STOP
```

```
END IF
```

```
1000 FORMAT(3I8)
```

13. WRITE

```
WRITE ([unit=]unit#, [fmt=]fmt [,iostat==ios] [,err=err-lab]) &
      [var list]
```

- **iostat:** (*iostat*) ios is an integer variable name, which will be set to zero on success.
- **err=lab:** jump to label lab on error.
- No real end-of-file probs with writes

```
INTEGER :: ierr, I, J, K
```

```
WRITE (unit=8, fmt=1000, iostat=ierr), I, J, K
```

```
! WRITE (8, 1000, iostat=ierr) I, J, K      ! same as line above!
```

```
! WRITE (8, '(3I8)', iostat=ierr) I, J, K   ! same as 2 lines above!
```

```
IF (ierr .NE. 0) THEN
```

```
    PRINT *, 'WRITE statement returned error: ', ierr
```

```
    STOP
```

```
END IF
```

```
1000 FORMAT(3I8)
```

14. Closing files with the CLOSE statement

```
CLOSE ([unit=]# [,iostat=ios] [,err=error-lab] [,status=st])
```

- **iostat:** (*iostat*) ios is an integer variable name, which will be set to zero on success.
- **err=lab:** jump to label lab on error.
- **status=st:** st is one of:
 - 'keep': Keep file around on filesystem after closing. (default for files opened without status='scratch').
 - 'delete': delete file after close is complete (default for files opened with status='scratch').

Examples:

```
CLOSE (8, iostat=ios, err=100, status='delete')
```

```
CLOSE (15, err=200)
```

```
CLOSE (11)
```

15. Program to read records and print as table, part 1 of 2

```
program read_records
  INTEGER :: age, I, ierr
  CHARACTER (LEN=80) :: first_name, last_name, infile

  PRINT*, 'Enter the filename to read from:'
  READ *, infile
  OPEN (unit=10, err=100, file=infile, status='old', action='read')
  WRITE (*, 1000)
  WRITE (*, 2000)
  i = 1
  DO
    READ (10, *, err=300, end=50) first_name
    READ (unit=10, fmt=*, err=300, end=200) last_name
    READ (fmt=*, unit=10, err=300, end=200) age
    WRITE(*, 3000) first_name, last_name, age
    i = i + 1
  END DO
50 CLOSE (10, iostat=ierr)
  IF (ierr .ne. 0) THEN
    PRINT 4000, trim(infile)
    STOP
  END IF

  PRINT *
  PRINT 5000, i
  STOP      ! end of normal execution
```

16. Program to read records and print as table, part 2 of 2

```
!
! Error labels jumped to from I/O READ errors
!
100 PRINT *, 'UNABLE TO OPEN FILE ', infile
    CLOSE (10)
    STOP
200 PRINT *, 'END-OF-FILE SEEN WHILE READING RECORD #', I
    CLOSE (10)
    STOP
300 PRINT *, 'INPUT ERROR IN RECORD #', I
    STOP

1000 FORMAT (' LAST NAME FIRST NAME AGE')
2000 FORMAT ('===== ===== ===')
3000 FORMAT(a10, 2x, a10, x, i4)
4000 FORMAT("ERROR CLOSING FILE '", a, "'")
5000 FORMAT('FINISHED PROCESSING ', i10, ' RECORDS WITHOUT ERROR')

END PROGRAM read_records
```