Fortran Basic I/O

Types of I/O Instructions

Input Instruction:	READ
Output Instruction:	WRITE
Data Arrangement:	FORMAT
Unit/File Information:	OPEN, CLOSE

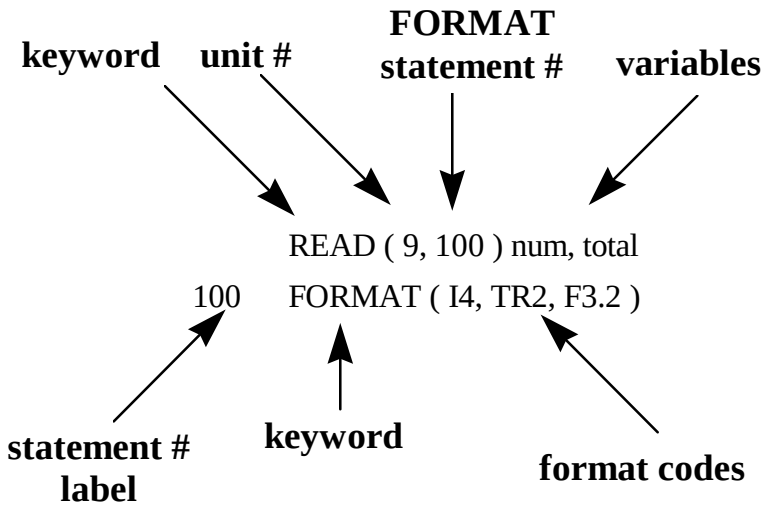
OPEN Statement

OPEN (9, FILE = 'P2.dat') *input file*

OPEN (10, FILE= 'P2.rst') *output file*

The open statement associates a file name and its attributes with a device unit number

READ Statement



WRITE Statement

The WRITE statement has an analogous syntax:

```
      WRITE (10, 200) num, total
200   FORMAT ('1', I5, TR3, F4.2)
```

Unit #'s: **9** - input for CS 1014 **10** - output for CS 1014

READ & WRITE Statement Requirements

Every* READ and WRITE must have both:

1) a unit number

2) a format statement number

Each read statement causes the computer to begin reading from the first column of the next input line

READ Example:

```
READ (9,100) x          2 lines
READ (9,101) y, z       3 values
```

The same rules apply for most write statements.

WRITE Example:

```
WRITE (10, 200) a       3 lines
WRITE (10, 201)         2 values
WRITE (10, 202) b       1 blank line
```

* unless defaults are used. We'll take a look at that alternative when we get to slides 3.10 and 3.11

Integer Input Codes

I - used for integer data only

I format code (input)

Iw where w is an integer >= 1, that tells how many columns to read

(Recall that in these notes that a beta β in the data file represents a blank space.)

Example:

```
      INTEGER :: j, k
      READ ( 9 ,100 ) j , k
100   FORMAT ( I3 , I4 )
      •    •    •
```

Data File Columns	(IN COMPUTER MEMORY)	
12345678901234	j	k
|β32ββ128β	32	12
|32ββ15012	320	150
|-41ββ7235	-41	72
|2.1	*** error ***	
	input type mismatch	

Results are shown for executing the READ statement upon each line with the same FORMAT statement.

Blank spaces are READ in as zeroes.

Real Input Codes

F used for REAL data only

F format code (input)

Fw.d where w and d are both integers, with the restrictions that
w >= 1 & d >= 0 & w >= d;

w # of total columns used for the width
d # of digits to right of decimal point
d is called the precision of the displayed value

Example:

```
      REAL:: a, b, c
      READ ( 9 , 110 )  a , b , c
110   FORMAT ( F5.2 , F7.3 , F3.0 )
      •     •     •
```

```
 line 12345678901234567      a       b          c
  1  |12.34ββ-.65987.ββ    12.34   -.659       87.0
  2  |ββ930ββ814ββ1234β      9.3    81.4      123.0
     |
```

If no decimal point is on data line then the **d** part of the format code is used to locate decimal point.
This is termed an **IMPLIED** decimal point.

If decimal point occurs on data line, it over-rides format code.

ex: if line 1 above was 1.234 then a is 1.234 even though the input format code for a is F5.2

Integer Output Codes

I format (output)

value is right justified in field

space must be allocated (column) for the negative sign (if needed)

Example:
```
INTEGER :: jmp, count, len
      :
WRITE ( 10 , 200 ) jmp , count , len
200   FORMAT ( TR1 , I2 , TR2 , I3 , I5 )
```

(SCREEN or FILE) (INSIDE COMPUTER)

columns variables

```
0 12345678901234  jmp    count   len
 |87  -23   15     87     -23      15
 |
```

Note: If print field is too small for value being printed, then field is filled with ******* (asterisks)

Example: Assume m is an integer variable

```
    WRITE(10, 210) m
210 FORMAT (TR1,I2)
```

```
m
---
105
```

```
0 12345
 |**
 |
```

execution (run-time) error

Real Output Codes

F format code (output)

Note: decimal point is always printed

The field width must allow space for the sign if value is negative.
At least one digit is always printed before decimal point.
Values are right-justified in the field.

Example: Assume alpha and beta are real variables

```
        WRITE ( 10 , 240 ) alpha , beta
240     FORMAT ( TR1 , F4.1 , TR2 , F5.2 )
```

(SCREEN or FILE) columns

```
0 123456789012345
 |23.5    3.65
```

(INSIDE COMPUTER) variables

```
alpha     beta
 23.5     3.65
```

TR1 in 240 above could have been ' ' (more on this later).

Note: Decimal in format code DICTATES (i.e. 'over-rides') placement of the decimal point in the OUTPUT).

Examples:

Assume a = 12.34 & b = 32.86

```
format  output
code  1234567890
F5.2 |12.34
F6.2 |β12.34
F8.3 |ββ12.340
F6.1 |ββ12.3
F4.2 |****
```

```
format   output
code    1234567890
F6.1   |ββ32.9
F3.0   |33.
```

Note: automatic rounding

Input/Output Codes

TR (tab right) format code (input/output)

TRw where w is an integer >= 1

Input

The next character to be read begins after w positions to the right of the current position.

Note: the columns are ignored regardless of whether they contain spaces or data.

Example: Assume i is an integer and z is a real variable

```
     READ (9,150) i, z
150  FORMAT (TR2, I3, TR1, F4.1)

          columns      variables
      12345678901234     i   z
     |458973β325681     897 32.5
     |
```

Output

The w denotes how many columns to move to the right on the printer/display line from the current WRITEing position.

Other similar format code or edit descriptors are: Tw (absolute Tab), TLw (Tab Left), and wX (Skip).

Note: TRw format code is identical with wX format code. However, the essential Lahey Fortran 90 (ELF90) does not support wX format code.

A format code (input/output)

Character Code Data

(more later)

Labelling Output

Literal Labels

character strings, (symbols enclosed in single quotes), may be included in a format code list.

all characters between the quotes are output verbatim

(i.e. treated literally NOT interpreted as format codes)

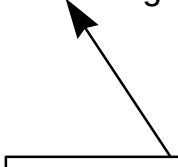
Example:
```
      REAL :: average_temp
            :
      WRITE (10,205) average_temp
205   FORMAT (TR1, 'Average temp is  ', F4.1,
     !' degrees.')
```

```
0 12345678901234567890123456789012345678901234567890
 |Average temp is 13.4 degrees.
 |
 |
```

Note: In a format statement, never continue a string past the end of a line.

Always break the string up into two shorter strings, putting the second string on the next line. Remember to put an ampersand (&) at the end of the line to indicate continuation to the next line.

```
FORMAT(' ', 'This is an example of a very', &
       ' long string?')
```



Without the space at the start of the second string the words would be printed without spacing.

Misc Output Info

REPEAT SPECIFICATIONS

Any series of adjacent format codes may be grouped together with parenthesis and preceded by an integer denoting the number of times the codes should be repeatedly used.

FORTRAN uses this convention simply for compaction of expression & to reduce typing.

Examples:

```
105 FORMAT (TR1, I2, I2, I2)
105 FORMAT (TR1, 3I2)

200 FORMAT (TR1, F3.1, TR2, F3.1, TR2, I6)
200 FORMAT (TR1, 2(F3.1,TR2), I6)
```

WRITE WITHOUT A VARIABLE LIST

Used to create output headings.

```
      WRITE ( 10, 206 )
206   FORMAT ( ' ', 'This program number 7.')
```

Note: NO variable list with WRITE and
NO format codes in the FORMAT statement

LIST-DIRECTED Input

Allows default device & free-format Input.

The standard method of READ input requires values to previously exist in a data file.

```
READ (9,100) a,b
```

The Input device number (9) can be replaced with an asterisk to invoke List-Directed INPUT.

```
READ (*, 100) a, b
```

The above read accepts data from the default device (usually the keyboard).

The FORMAT statement (100) can be replaced with an asterisk to accept free-format input.

```
READ (9, *) a, b
```

The above read takes data from unit 9 BUT unformatted (separated by blanks).

Asterisks can be used simultaneously for the input device & format statement number.

```
READ (*,*) a, b
```

Accepts unformatted input from the default device (keyboard).

LIST-DIRECTED Ouput

Allows default device & free-format output.

The standard method of WRITE output requires values to be placed in a data file.

```
WRITE (10, 200) a, b
```

The output unit device number can be replaced with an asterisk

```
WRITE (*,200) a, b
```

writes values to a default device (usually video display)

The format statement number can be replaced with an asterisk.

```
WRITE (10, *) a, b
```

write values to unit 10, BUT use a default formatting scheme (for example: reals may use F10.8, etc.), compiler dependent.

Asterisks can be used simultaneously for the output device & format statement number.

```
WRITE (*,*) a, b
```

Produces unformatted output on the default device.

NOTE: Using an ***** in place of a unit number and/or format statement number will **create output** that will **NOT** be **graded** (properly) by the automatic grading system. Use * carefully to do interactive I/O or unformatted I/O.

List-Directed I/O should only be used for testing & debugging purposes.

Carriage-Control Characters

Carriage-Control Characters (CCC) are used in Fortran for output page & line spacing.

CCCs are used ONLY in conjunction with WRITE statements

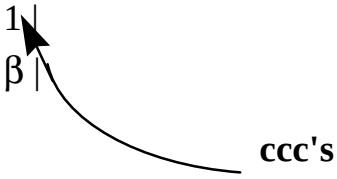
First character of each OUTPUT line is used to control movement of paper in FORTRAN

CCC Table

character	effect
' '	(blank space) single space down the page
'0'	(zero) double space down the page (1 blank line, then ...)
'1'	(one) form feed (print at top of next page)
'+'	(plus) overprint (previous line) with this line (not sup.)
'-'	(minus) triple space (not widely supported)

The first column with ccc's is designated as column 0

```
0 12345678901234567890
1
β
```



ccc's

Carriage-Control Characters

Examples

WRITE (10, 204) num { Assume that num is an integer and is 23 }

```
                          disk     printed
                          file     output
                          0 12345   12345

204  FORMAT (' ', I3)    |bb23    |b23
204  FORMAT (TR1, I3)    |bb23    |b23
204  FORMAT ('0', I3)    |0b23    |b23  <after blank line>
204  FORMAT ('1', I3)    |1b23    |b23  <at top of page>
204  FORMAT (I3)         |b23     |23
204  FORMAT ('1')        |1       |     <at top of page>

     WRITE(10,205)       |        |
                         |        |
205  FORMAT ('1fred')    |1fred   |fred <at top of page>
                         |        |
205  FORMAT (' ', 'fred''s')
                         |bfred's|fred's
                         |        |
205  FORMAT (' ','''wow''')
                         |b'wow' |'wow'
                         |        |
```

What gets printed for the example below?

```
     num = 123
     WRITE (10,206) num
206  FORMAT (I3)
```

Example with File I/O and Formatting

The following program reads two values from an input file, calculates their product, and prints results to an output file:

```
PROGRAM input_output
  IMPLICIT NONE
! Declare a real variable
  REAL :: height, width, area

  OPEN ( 9, FILE = 'ifile.txt')
  OPEN (10, FILE = 'ofile.txt')

! Read values for height and width from ifile.txt
  READ ( 9, 100) height, width

! Calculate the area
      area = height * width

! Print conclusions to ofile.txt
  WRITE (10, 200) height, width, area

! Close the files
  CLOSE ( 9)
  CLOSE (10)

  STOP
100  FORMAT (F5.2, TR1, F5.2)
200  FORMAT (TR1, F5.2, ' by ', F5.2, ' has area ', F8.2)
  END PROGRAM input_output
```

Given the input file:

```
72.54 14.17
```

The program produces an output file containing:

```
 72.54 by 14.17 has area  1027.89
```