

Goals for This Lecture:

- Learn the use of I/O format descriptors

Formats in WRITE statements

- The second asterisk in the WRITE statement can be replaced by a FORMAT string that specifies how the data should be output

- Example:

```
write(*,'(1x,i4,2x,f10.2)') i,x_position
```

- This format specifies that the output should skip 1 space from the beginning of the line (1x), should output the integer i using an integer four digit field (i4), skip 2 spaces (2x), and output the real variable x_position as a decimal 10 digit field with 2 digits to the right of the decimal place(F10.2)

- This format string could be placed in a FORMAT statement that is labeled with a numerical value from 1 to 5 digits

- Example:

```
write(*,1000) i,x_position  
1000 format(1x,i4,2x,f10.2)
```

- The format string could be placed in character variable

- Example:

```
character(len=30) fmt1='(1x,i4,2x,f10.2)'  
write(*,fmt1) i,x_position
```

Format Descriptors in WRITE statements

- The items in the FORMAT string are called format descriptors
- Format descriptors control the position and manner in which data is displayed
- Format descriptors fall into four categories:
 - Position descriptors that control the vertical positioning of the data
 - Position descriptors that control the horizontal positioning of the data
 - Edit descriptors that control the output format of the data
 - Repetition counts that control the repetition of portions of the format

The I edit descriptor (Integer Output)

– Form: *rIw* or *rIw.m*

r is the repeat count

w is the field width (number of spaces out will occupy)

m is the minimum number of digits to be displayed

– Integers are right justified in the field

– If the integer is larger than the field width, then the field is filled with asterisks

– Example:

```
write(*, '(3i4,i4.3)') 1535,37,37,37
```

```
1535  37  37 037
```

The F edit descriptor (Real Output)

- Form: `rFw.d`
 - `r` is the repeat count
 - `w` is the field width (number of spaces out will occupy)
 - `d` is the number of digits to be displayed to the right of the decimal point $< w-2$ (must leave room for decimal & minus sign)
- Real values are right justified in the field
- If necessary, the number is rounded off before being displayed
- If the value is larger than the field width, then the field is filled with asterisks
- Error in book: If displayed number has more digits than the internal representation then extra zeros are appended to the right of the decimal point (**Not true! Compiler dependent**)
- Example:

```
write(*,'(3f6.3,f9.6)') 1535.0,-1./3.,37.0,3.14
*****-0.333 37.000 3.140000
```

The E edit descriptor (Real Output)

- Form: $rEw.d$
 - r is the repeat count
 - w is the field width (number of spaces out will occupy)
 - d is the number of digits to be displayed to the right of the decimal point $\leq w-7$ (must leave room for decimal, minus sign, E, exponent sign, and two digit exponent field)
- Mantissa is displayed as a value between 0.1 & 1.0
- If necessary, the number is rounded off before being displayed
- If the value is larger than the field width, then the field is filled with asterisks
- Example:

```
write(*, '(2e10.3,e10.2)') 1535.0e23,-1./3.,37.e-22
0.153E+27-0.333E+00 0.370E-20
```

The ES edit descriptor (Real Output) Scientific Notation

– Form: *rESw.d*

r is the repeat count

w is the field width (number of spaces out will occupy)

d is the number of digits to be displayed to the right of the decimal point $\leq w-7$ (must leave room for decimal, minus sign, E, exponent sign, and two digit exponent field)

– Mantissa is displayed as a value between 1.0 & 10.0

– If necessary, the number is rounded off before being displayed

– If the value is larger than the field width, then the field is filled with asterisks

– Example:

```
write(*, '(2es10.3,es10.2)') 1535.0e23,-1./3.,37.e-22
```

```
1.535E+26-3.333E-01 3.700E-21
```

The L edit descriptor (Logical Output)

– Form: *rLw*

r is the repeat count

w is the field width (number of spaces out will occupy)

– Logicals are displayed as a T or F

– Values are right justified in the field

– Example:

```
write(*, '(2L3)') .true., .false.
```

T F

The A edit descriptor (Character Output)

– Form: *rA* or *rAw*

r is the repeat count

w is the field width (number of spaces out will occupy)

– Values are right justified in the field

– If the length of the value is longer than the field width the string is truncated

– Example:

```
write(*, '(3A2)') 'AB', 'C', 'DEF'
```

```
AB CDE
```

The T position descriptor

– Form: **T***c*

c is the column number

– The next edit descriptor field will start in column *c*

– Example:

```
write(*, '(T4,3A2)') 'AB','C','DEF'
```

```
AB CDE
```

The X position descriptor

– Form: ***n*X**

n is the number of spaces to skip

– Skips ***n*** spaces between the previous edit descriptor and the next edit descriptor

– Example:

```
write(*, '(A2,2X,A2,2X,A2)') 'AB', 'CD', 'EF'
```

```
AB  CD  EF
```

The X position descriptor

– Form: ***n*X**

n is the number of spaces to skip

– Skips ***n*** spaces between the previous edit descriptor and the next edit descriptor

– Example:

```
write(*, '(A2,2X,A2,2X,A2)') 'AB','CD','EF'
```

```
AB  CD  EF
```

Vertical control

- On older line printers characters in column 1 can be used to control the vertical spacing of the output
- These characters are:
 - + to avoid advancing to the next line
 - 0 to skip a line
 - 1 to skip to next page
 - A blank starts a new line
- While these fields only work on devices that support vertical carriage control it is good programming practice to start your output in column 2 or beyond using a T2 position descriptor to avoid any spurious behavior in the vertical spacing of your output

Vertical (Carriage) control

- On older line printers characters in column 1 can be used to control the vertical spacing of the output
- These characters are:
 - +** to avoid advancing to the next line
 - 0** to skip a line
 - 1** to skip to next page
 - A blank starts a new line
- While these fields only work on devices that support vertical carriage control it is good programming practice to start your output in column 2 or beyond using a T2 position descriptor to avoid any spurious behavior in the vertical spacing of your output

The / (slash) descriptor

- The slash descriptor can be used to start a new line during a WRITE operation
- Example:

```
write(*,'(t2,a2,2x,a2,/,t2,a2,2x,a2)') 'ab','cd','ef','gh'
```

```
ab cd
```

```
ef gh
```

- The slash descriptor does not have to be separated from other descriptors by commas but it is good practice to do so

The \$ descriptor (non-standard but widely supported)

- The \$ descriptor can be used to avoid printing a carriage return at the end of a line of output
- Example:

```
write(*, '(t2,all,$)') 'Enter data:'
```

Enter data:

- Useful for setting up prompts to user
- Non-standard but almost every compiler supports this descriptor

Repeating groups of format descriptors

- Groups of descriptors can be enclosed in parenthesis and preceded by a repeat count
- Example:

```
write(*,'(4(1x,f3.1))') 1.1,2.1,3.1,4.1
```

```
1.1 2.1 3.1 4.1
```

- Useful for outputting a lot of variables with uniform spacing between columns

Use of formats in WRITE statements

- Formats descriptors are applied in left to right order
- If a repetition count precedes a descriptor (or group of descriptors enclosed by parenthesis) that descriptor (or group of descriptors) is applied the specified number of times before applying the next descriptor in the format
- If the number of data items is less than the number of edit descriptors the write statement terminates when it reaches the first descriptor with no corresponding data item
- If the number of data items is greater than the number of edit descriptors a new line is started after the last data item corresponding to an edit descriptor is printed. Subsequent data items are printed on a new line applying the format from the beginning (Note: your textbook is in error in its explanation of rule 5 on page 187)
- Example:

```
write(*,'(2(1x,i4.4),1x,i3.3))' ) 333,333,333,333  
0333 0333 333  
0333
```

Use of formats in READ statements

- Avoid formatted READs if you can possibly do so!
- Use free-format READs whenever possible
 - Free-format READs are more flexible and allow for slight differences in formatting of data
 - Let the compiler do the work for you!
- Formatted READs are highly susceptible to errors
 - Formatted READs are very rigid
 - If you are off by one column in your format you can get an unwanted value
- If you absolutely must use formatted reads consult your book for details on how to do so

Reading Assignment

- Read Sections 5.5-5.6