Chapter 5
Input and Output

Formatted output:

`PRINT format-specifier, output-list`

where `format-specifier` is one of the following:

1. *
2. A character constant or variable or expression whose value specifies the format of the output
3. The label of a FORMAT statement

If the output list is omitted, a blank line is displayed.

The asterisk * means the default format which is machine-dependent.

The method 2 uses:

`"(list of format description)"` or

`'(list of format description)'`

Example:

```
PRINT '(1X, I3)', N

PRINT '(1X, 2I4, 2F6.3, F8.3)', In, Out, A, B, C
```

The method 3 uses FORMAT statement

`label FORMAT (list of format description)` where the label is a positive integer less than 99999.

Example:

```
INTEGER :: Number = 3, M = 0, L = 5378, Kappa = -12345
PRINT 30, Number, M, L, Kappa
PRINT 31, Number, M, L, Kappa

30 FORMAT (1X, 2I5, I7, I10)
31 FORMAT (1X, 2I5.2, I7, I10.7)
```

`1X` means a normal space. It is better to put it first, because some compiler uses first character to control space.

The I descriptor is used to format integers.

`nIw.m`

`I` indicates a format of an integer which is right-justified. `w` indicates the wide of the output and `m` indicates the minimum output digits. `n` means the format repeat *n* times.

The output of the previous slide is:

```
    3     0    5378      -12345
   03    00    5378    -0012345
```

Real output using `F, E, ES, EN` descriptors which is also right-justified.

`Fw.d`

Real data in decimal notation with wide `w` and `d` specifies the number of digits to the right of the decimal point. Note that $w \geq d + 3$, because of the space of sign, zero and decimal point.

`Ew.d Ew.dEe`

Real data in exponential notation. `e` specifies the number of digits in exponent. Note $w \geq d + 7$ or $w \geq d + e + 5$.

`ES EN`

Similar to `E`, the mantissa is at least 1 but less than 10 for `ES`. The exponent is constrained to be a multiple of 3.

In both integer and real format, if the data needs more space than the space provide by the format, the computer will display

```
***********
```

For examples:

```
INTEGER :: N = 123456
REAL :: R = 2.34E-12
PRINT '(1X, I4, E8.2)', N, R
PRINT '(1X, I7, F6.2)', N, R
```

What are the output?

Character output

May include character strings in the list of descriptor of format.

`PRINT '(1X, "X=", F6.2, " Y =", F6.2)', X, Y` or

```
PRINT 70, X, Y
70 FORMAT(1X, "X=", F6.2, " Y =", F6.2)
```

`A Aw`

Using `A`, the computer will decide the field width by the character string. `Aw` specifies the width of the field. In this case, the display is right-justified. If the width is smaller than the string, then only left-most w characters will be displayed.

Position descriptors `X T`

`nX` causes n blanks inserted.

`Tc` causes the next output field to begin at the specific position `c`.

```
PRINT 75, "John Q. Doe", "CPSC", Number
75 FPRMAT(1X, A11, T16, A4,2X,I3)
```

The output will be:

```
 John Q. Doe   CPSC  141
```

The `/` descriptor causes the output to begin on a new line. It is not necessary to use a comma to separate a slash from other descriptors

```
PRINT 88, "Values", n, m, b
88 FORMAT(1X, A, 3/ 1X, I6,I7 // 1X E15.7)
```

The output: $(n = 5173, m = 7623, b = 37.555)$

```
 Values



   5173   7623


  0.3755500E+02
```

If the values of all the items in the output list have been displayed before all the descriptors have been used, scanning of the format specifier continued until

1. The right parenthesis signalling the end if the list of format descriptors.
2. An `I, F, E, ES, EN,A,L,G, B, O` of `Z` descriptor
3. A colon

Example:

```
PRINT '(1X, I5, 3I6)', M, N
PRINT '(1X, F5.1, F7.0, F10.5)', A, B
PRINT '(1X, 5(" Item is", A10))', "Bumper", "Headlight"
PRINT '(1X, 5(: " Item is", A10))', "Bumper", "Headlight"
```

The output will be: $(M = 7623, N = 5176, A = 617.2, B = 29.25)$

```
  7623  5176
 617.2    29.
   Item is     Bumper  Item is Headlight  Item is
   Item is     Bumper  Item is Headlight
```

If the list of descriptors is exhausted before the output list is, a new line of output is begun, and the format specifier or part of it is rescanned.

- If there are no internal parentheses within the format specifier, the rescanning begins with the first descriptor.
- If the format specifier contains internal parentheses, rescanning begins at the left parenthesis that matches the next-to-last right parenthesis.

Example:

```
PRINT '(1X, 2I6)', N, N+1, N+2, N+3, N+4
PRINT 100, M, A, N, B, N+1, B+1.0, N+2, B+2.0
100 FORMAT(1X, I5, F10.3 / (1X, I10, F12.2))
```

Output:

```
   5173  5174
   5175  5176
   5177
  7623   617.200
       5173       29.25
       5174       30.25
       5175       31.25
```

Formatted input

```
READ format-specifier, input list
```

The format-specifier is similar to that of output (PRINT) except it cannot contain character strings and the colon.

The formatted input is more useful for input from a file. Using formatted input for keyboard input might cause problems, because it is not easy to follow the format to type.

Integer input

```
INTEGER :: I, J, K
READ '(I6,I4, I7)', I, J, K
PRINT *, I, J, K
```

If the input is:

```
-123   45  678
```

the out put is `-123 45 678`

If the input is:

```
-123  45 678
```

the output is `-123 456 78`

What if the input is `12345612341234567`?

Real input

- The numbers may be entered without decimal points.
- The decimal point may be entered as part of the input value.

```
REAL :: A, B, C, D, E
READ '(F3.2, 2F3.1, F3.3, F6.3)', A, B, C, D, E
PRINT *, A, B, C, D, E
```

If the input is `615-19750182625327` the output is:

```
6.15 -1.9 75.0 0.182 625.327
```

But if the input is `6.15 -1.9 75.0 0.182 625.327`, then output will be

```
 ******  FORTRAN RUN-TIME SYSTEM  ******
 Error 1081:  unexpected character in real field
 Location:  the READ statement at line 8 of "fmt_input.f90"
 Unit:  *
 File:  standard input
Abort (core dumped)
```

If the input is `3.2 21.1 222`, then output is:

```
3.2 2.1 0.1 0.222 0.0E+0
```

Character input

The output of character strings are depended on the length of the string and the output format. Read format different from the string length may cause problems.

```
CHARACTER(6) :: S1, S2
READ '(2A6)', S1, S2
PRINT *, S1, S2
READ '(A2,A12)', S1, S2
PRINT *, S1, S2
```

If the both inputs are: `inputs two strings`, the output will be:

```
inputs two s
```

and

```
input string
```

`X, T` can be used to skip input characters. Sometimes this is useful for input from files.

If the input is `I = 4     J = 56   K = 137`

we can use the following statements

```
READ '(3X, I2, 6X, I3, 5X, I4)', I, J, K
```

or

```
READ '(T4, I2, T12, I3, T20, I4)' I, J, K
```

A new line of data is required each time a `READ` statement is executed of whenever a slash is encountered in the format specifier. This is also useful for file input.

```
REAL :: Amount, Rate
READ '(/ F6.0 3/ F4.0)', Amount, Rate
```

can read the following data

```
Amount to be produced
585.00
Reaction rate
(This assumes constant temperature)
5.75
```

The general `WRITE` statement is:

```
WRITE (control-list) output-list
```

The control-list may include selected from:

1. A unit specifier: `UNIT = unit-specifier` or `unit-specifier`. If UNIT is omitted, it must be the first item.
2. A format specifier: `FMT = format-specifier` or `format-specifier`. If FMT is omitted, it must be the second item.
3. An `ADVANCE =` clause (the clause is either `YES` or `NO`).
4. Other items that are useful in processing file.

Examples:

```
WRITE (6, *) Gravity, Weight
WRITE (UNIT = 6, FMT = *) Gravity, Weight
WRITE (Output_Unit, *) Gravity, Weight
WRITE (UNIT = Output_Unit, FMT = *) Gravity, Weight
WRITE (*, *)  Gravity, Weight
```

The `ADVANCE` is used to specify whether output should advance to a new line after the current output has been completed. The default is `YES`.

The general `READ` statement

```
READ (control-list) input-list
```

The control-list includes selected from (most are similar to `WRITE`):

1. A unit specifier
2. A format specifier
3. An `IOSTAT =` clause (used for input from file)
4. Other items that are useful in processing file.

Application involving large data or saved data, a file is used to store data. Before a file can be used in a Fortran program, the file must be open first, including connecting a unit number and several items of information.

```
OPEN (Open-list)
```

The open list includes several things.

1. A unit number.
2. `FILE = character-expression` connect the file name to the unit number.
3. `STATUS = character-expression`, the expression is one of `"OLD" "NEW" or "REPLACE"`.
4. `ACTION = io-action`, the action is one of `"READ" "WRITE" or "READWRITE"`.
5. `POSITION = character-expression`, the expression is one of `"REWIND" "APPEND" or "ASIS"`.
6. `IOSTAT = status-variable`, the variable is assigned a zero if the file open successfully or a positive value otherwise. `IOSTAT` also can be used for file input and output. We will discuss later.

Example:

```
INTEGER :: OpenStatus
CHARACTER(12) :: FileName
WRITE (*, '(1X, A)', ADVANCE = "NO") &
   "Enter name of data file: "
READ *, FileName
OPEN (UNIT = 12, FILE = FileName, STATUS = "OLD", &
   ACTION = "READ", POSITION = "REWIND", &
   IOSTAT = OpenStatus)
```

The following statement can be used:

```
IF (OpenStatus > 0 STOP "** Cannot open file **"
```

Example:

```
OPEN (UNIT = 13, FILE = "REPORT", STATUS = "NEW", &
   ACTION = "WRITE", IOSTAT = OpenStatus)
IF (OpenStatus > 0) STOP "** Cannot open file **"
WRITE (13, '(1X, I3, F7.0, F10.2)') Code, Temp, Press
```

These statements open a new file named REPORT and write three values to that file according to the format.

There are some other clauses may be used:

- `BLANK =` clause, specifying how blanks in numeric fields are interpreted.
- `DELIM =` clause, specifying whether character strings written to the fild are to be enclosed in delimiters.
- `ERR =` clause, specifying a statement to be executed if an error occurs.
- `FORM =` clause, specifying whether the file is formatted or unformatted.
- `PAD =` clause, specifying whether character inputs are to be padded with blanks.
- `RECL =` clause, specifying the record length for a direct access file.

`CLOSE (close-list)` statement closes the file and disconnects a file from its unit number.

The close-list must include a unit specifier and may include other items such as `IOSTAT, ERR, STATUS`.

After a file is opened, the file is connected to a unit number. This number can be put as the number of device in `READ, WRITE` statement, if the input and output are using that file. The format discussed previously can be used for file input and output.

`IOSTAT` will assign the status-variable as:

1. A positive value if and input error occurs.
2. A negative value if the end of data is encountered but no input error occurs
3. Zero if neither an input error not the end of data occurs.

Example:

```
DO
  READ (12, *, IOSTAT = InputStatus) Code, Temperature, Pressure
  IF (InputStatus < 0) EXIT
  IF (InputStatus > 0) STOP "** Input error **"
  Count = Count + 1
  ...
  Sum = Sum + Pressure
END DO
```

Sometimes it is necessary to reposition a file so that data values that have been already read can be read again.

`REWIND unit-number` will reposition a file at its begging .

`BACKSPACE unit-number` will reposition a file at the beginning of the preceding line.

If the file is at its initial point, these statements have no effect.

To reposition within a line, we can use `X, T` descriptors.

To skip some lines, we can use `/` descriptors.

Example: Suppose that a device monitoring a process records time, temperature, pressure and volume and stores data in a file as follows:

```
1200034203221015
1300038803221121
1400044803241425
... ...
2200088903303186
```

where time is an integer using positions 1-4, temperature, pressure and volume are real numbers, each of them using 4 positions. The decimal point is at the position between the third and fourth digits (which is omitted in records).

A program is to be requested to read the values for the temperature and volume, display these values in tabular form and display the average temperature and the average volume.

Analysis and design

algorithm

pp. 111