Computer Science 1510

Lecture 6

Lecture Outline

- Input/Output
- Formatted input/output

Input and Output

- The computer accepts input data from the keyboard or a file in the form of a *stream*.

- A text stream is a sequence of characters divided into lines, with each line consisting of a sequence of characters followed by a newline character.

- To input the number 352 from the keyboard the following would occur:
  - Type the '3' key. This digit is placed in a location in memory called a *buffer*.
  - Type the '5' key. This digit is placed in the buffer behind the '3'.
  - Type the '2' key. This digit is placed in the buffer behind the '5'.
  - Finally press the 'Enter' key which is the signal to the computer that you have finished entering the string. The string '352\n' is then given to your program.

Input and Output

- A buffer is an empty location in RAM (several blank bytes) that is used for temporary storage (such as accepting input strings as above).

- A buffer of size 10 could be represented as the following:

0	1	2	3	4	5	6	7	8	9

  Each of the 10 *cells* in the buffer consist of a byte of memory and a label (here labels 0-9).

- If we enter the name 'John' using the keyboard the buffer fills up starting in cell 0.

0	1	2	3	4	5	6	7	8	9
J	o	h	n						

- The buffer is normally emptied or cleared when the user types the Enter key following the input of a string.

Input and Output

- Streams are also used when outputting information to the screen or to a peripheral device such as a printer.

- Each character is stored as an 8-bit byte. Memory is organized into 8-bit bytes, and peripheral devices work with 8-bit bytes.

- Peripheral devices usually have built-in character tables (such as the ASCII table).

- This makes it easy to pass character data between devices with little or no conversion required.

Input/Output

- There are two main input/output statements in Fortran.

- The `READ` statement is used to obtain information, that is, to input data into a program.

- The `WRITE` statement is used to output data from the program.

- Both the `READ` and the `WRITE` statement can be used to input/output information from/to the user or from/to a file.

  We will initially look at user interaction, and later discuss reading from/ writing to a file or files.

- The simplest form of each of these statements is called the *list-directed form*.

  This is the form we have already seen.

Fortran statements: WRITE

- The `WRITE` statement is used to print information to the screen or to a file.

- The list-directed form of the `WRITE` statement assumes certain default formatting rules.

- Syntax:

```
WRITE(*,*) output-list
```

  where `output-list` is a single expression or list of expressions separated by commas to be printed to the screen.

- Each `WRITE` statement prints a new line of output.

  If the `output-list` is omitted, a blank line will be printed.

- Example:

```
WRITE(*,*) 'The average value is',average
```

  where `average` is a variable.

Fortran statements: READ

- The `READ` statement is used to obtain (*read*) information from the user or from a file during run-time.

- The list-directed form of the `READ` statement reads values in the order in which they are input.

- Syntax:

  ```
  READ(*,*) input-list
  ```

  where `input-list` is a single variable or list of variables separated by commas to which the input data is to be assigned.

- If the `input-list` is omitted, a line will be skipped.

- When a `READ` statement is encountered during run-time, execution is suspended until the user has provided the necessary input. Execution then automatically resumes.

Fortran statements: READ

- A new line of data is processed each time a `READ` statement is executed.

- If there are more values on a line than there are variables in the `input-list`, the first values are assigned in order, and the remaining values are ignored.

- If there are fewer values entered on a line than there are variables in the `input-list` then successive lines are processed until all variables in the `input-list` have been assigned values.

- Example:

```
WRITE(*,*) 'Enter values for the initial amount,&
          & time, and halflife'
READ(*,*) init, time, halflife
```

- Input data should be separated by commas, spaces, or newlines.

Example: READ and WRITE

```
PROGRAM Projectile
!-----------------------------------------------------------------------
! This program calculates the velocity and height of a projectile
! given its initial height, initial velocity, and constant
! acceleration. Identifiers used are:
!   accel    : vertical acceleration (m/sec/sec)
! Input:
!   initH    : initial height of projectile (meters)
!   initV    : initial vertical velocity (m/sec)
!   time     : time since launch (seconds)
! Output:
!   height   : height at any time (meters)
!   velocity : vertical velocity at any time (m/sec)
!-----------------------------------------------------------------------

  IMPLICIT NONE
  REAL :: initH, height, initV, velocity, time
  REAL, PARAMETER :: accel=-9.80665

! Obtain values for initH, initV, and time
  WRITE(*,*) 'Enter the initial height (m) and velocity (m/sec):'
  READ(*,*) initH, initV
  WRITE(*,*) 'Enter time in seconds at which to calculate height and velocity:'
  READ(*,*) time

! Calculate the height and velocity
  height = 0.5*accel*time**2 + initV*time + initH
  velocity = accel*time + initV

! Display velocity and height
  WRITE(*,*) 'At time', time, 'seconds'
  WRITE(*,*) 'the vertical velocity is', velocity, 'm/sec'
  WRITE(*,*) 'and the height is', height, 'meters'

END PROGRAM Projectile
```

Fortran statements: FORMAT

- So far we have seen only the simplest (list-directed) form of both the WRITE and READ statements.
- In list-directed form the WRITE statement prints information using default spacing, which is often not the nicest format.
- By default a READ statement that reads in two numbers requires a comma or a space between the numbers, and a WRITE statement prints variable values with tab spacing.
- This behaviour can be modified using a FORMAT statement which specifies exactly how information is input or output.
- Syntax:

```
label FORMAT layout-information
```

Fortran statements: FORMAT

- Each FORMAT statement is linked to an input or output statement via a `label` which can be any number from 1 to 99999.

- The `label` number replaces the second * in the corresponding input or output statement.

- FORMAT statements can be placed anywhere in a program AFTER the declaration statements.

- It is good practice to place a FORMAT statement immediately following the input or output statement to which it applies.

- The `layout-information` is a list of format descriptors separated by commas as discussed below.

- A single `FORMAT` statement can be used by any input/output statements that require the specified format.

Formatted output

- Syntax (to print to the screen):

```
WRITE(*,label) output-list
```

- Example:

```
    INTEGER :: a=2, b=3
    WRITE(*,10) 'The sum of a and b is',a+b
10  FORMAT(A40,I3)
```

- The format label here is `10`.

- The first argument `A40` specifies that a character string up to 40 characters long is to be printed, using exactly 40 spaces.

- The second argument `I3` specifies that an integer up to 3 digits is next to be printed, using exactly 3 spaces. Note that if the integer was 3 digits then there would be no space between the character string and the number, when printed.

Formatted output

- The output produced from this program is:

```
The sum of a and b is  5
```

  Note that the beginning of the 40 character string is padded with blank spaces (or right-justified).

- If the first argument was `A15` instead of `A40` the following output would result since the string to be printed is more than 15 characters long:

```
The sum of a an  5
```

  Note that it is now left-justified, and truncated.

- The format descriptors (`A40` and `I3`) specify the type and size of the data to be printed.

- Omitting the size on the `A` descriptor results in a character field equal to the width of the string being printed. In the above case:

```
The sum of a and b is  5
```

Format descriptors

Form(s)	Use
rIw or rI$w.m$	Integer data
rF$w.d$	Real data (decimal notation)
rE$w.d$ or rE$w.d$Ee	Real data (exponential notation)
rES$w.d$ or rES$w.d$ESe	Real data (scientific notation)
rLw	Logical data
rA or rAw	Character data
nX	Horizontal spacing
Tc	Tab spacing (to column c)
'. . . '	Character strings (output as is)
$r/$	Vertical spacing
( )	To repeat blocks of descriptors

- r = repetition indicator (number of such fields). (If not indicated, a value of 1 is assumed).
- w = total width of field.
- m = minimum number of digits.
- d = number of digits to the right of the decimal.
- e = number of digits in the exponent.
- n = number of character positions.

Displaying integer data: The I descriptor

$$r\texttt{I}w \qquad r\texttt{I}w.m$$

- `I` is used to denote integer data.
- w is an integer specifying the width of the field (number of spaces required), including the sign.
- r is a repetition indicator, indicating the number of such fields. E.g., `3I2` is equivalent to `I2,I2,I2`.
- m is the minimum number of digits to be displayed (often omitted).
- Integer data is right justified in the field.
- Example:

```
      INTEGER::i,j,k
      i=2389; j=1; k=865
      WRITE(*,10) i,j,k
   10 FORMAT(3I4)
```

- Output:

```
2389   1 865
```

Displaying real data: The F descriptor

$$r\mathtt{F}w.d$$

- `F` is used to denote real (floating-point) data (in decimal notation).
- w is an integer specifying the total width of the field, including the sign, the decimal point, and any leading zeros ($w \geq d + 3$).
- d is an integer specifying the number of digits to the right of the decimal. (Values with more than d digits after the decimal are rounded).
- r is a repetition indicator, indicating the number of such fields.
- Real data is right justified in the field.
- For both integer and real data, if the specified width of the field is not sufficient for the given value, *'s will be displayed.

Displaying character data: The A descriptor

$$r\mathtt{A} \qquad r\mathtt{A}w$$

- `A` is used to denote character data.
- w is an integer specifying the width of the field (number of characters).
- r is a repetition indicator, indicating the number of such fields.
- In the first case, where a width is not specified, the width of the field is determined by the length of the string to be displayed.
- In the case where a field width is specified, and it is sufficiently wide to hold the string to be displayed, the string is right justified in the field.
- In the case where the width of the field is not sufficient for the given string, the leftmost w characters are displayed.

Formatting without FORMAT

- Input and output can be formatted by including format descriptors within the corresponding `WRITE` and `READ` statements.

- The format specifier should be written as `'(list-of-descriptors)'` in place of the second `*` (ie. in place of the `label`).

- Example:

```
WRITE(*,'(A,I3)') 'Age', years
```

  or

```
WRITE(*,FMT='(A,I3)') 'Age', years
```

Example 1: Format descriptors

```
PROGRAM Format_integers
! This program illustrates the use of format statements
   IMPLICIT NONE
   INTEGER :: m,n
   m=-12; n=375

   WRITE(*,*) 'Output without formatting:'
   WRITE(*,*) 'm=',m,'n=',n
   WRITE(*,*) 'Output with formatting:'
   WRITE(*,10) m,n
10 FORMAT ('m=',I4,X,'n=',I4)
END PROGRAM Format_integers
```

Output:

```
 Output without formatting:
 m=         -12 n=         375
 Output with formatting:
m= -12 n= 375
```

Example 2: Format descriptors

```
PROGRAM Format_reals
! This program illustrates the use of format statements
   IMPLICIT NONE
   REAL :: x,y
   x=15.2; y=-3.14159

   WRITE(*,*) 'Output without formatting:'
   WRITE(*,*) 'x=',x,'y=',y
   WRITE(*,'(A)') 'Output with formatting:'
   WRITE(*,'(A2,F7.2)') 'x=',x,'y=',y
   WRITE(*,'(A2,E9.2)') 'x=',x,'y=',y
   WRITE(*,'(A2,ES9.2)') 'x=',x,'y=',y
END PROGRAM Format_reals
```

Output:

```
 Output without formatting:
 x=   15.200000     y=  -3.1415901
Output with formatting:
x=  15.20
y=  -3.14
x= 0.15E+02
y=-0.31E+01
x= 1.52E+01
y=-3.14E+00
```

The ADVANCE clause

- In each of the examples that we have seen, the `WRITE` statement prints the `output-list` and moves to the next line on the screen.

- To suppress moving to the next line, we can use

  `ADVANCE="NO"`

  within the write statement. Use of the `ADVANCE` clause requires formatted output.

- Example:

```
WRITE(*,'(A)',ADVANCE="NO") 'Initial amount: '
READ(*,*) init
```

  will print:

```
Initial amount:
```

  to the screen and wait for the user to enter a value. The value entered will appear on the same line as `Initial amount:`

Formatted input

- Formatted input uses similar format descriptors as we have discussed for formatted output.

- A form of the READ statement is used to read in data in a particular format.

- Formatted input is generally only needed when reading data (usually from a file) that is in a predetermined format.

- We will postpone our discussion of formatted input until we consider file input/output.