Read Loop

What if it is not known how many data lines there are in the data file ? (assume more than one line)

We want to read a value from each line of the input file, do a calculation, write out a result and continue until we have reached the END-OF-FILE.

This can be done with a READ statement in a DO…END DO loop.

The READ statement may have an optional IOSTATclause that allows to check a variable for "end of file" contidion in an IF statement to exit from the DO…END DO loop.

Example

Note: the statements that comprise the loop are indented to denote that they are *under control* of the loop. Indent all loop statements.

```
      PROGRAM temperature
      IMPLICIT NONE
      REAL :: cvalue, fvalue
      INTEGER :: ios
            :
      READ (9,100,IOSTAT=ios) fvalue
100   FORMAT (F8.2)
      DO
         IF (ios < 0) EXIT          ! Exit at end of file
         cvalue = (5.0 / 9.0) * (fvalue - 32.0)
         WRITE (10, 200) fvalue, cvalue
200      FORMAT ('0', 'Fahr = ', F8.2, TR3, 'Cel = ', F8.2)
         READ (9,100,IOSTAT=ios) fvalue
      END DO
            :
      STOP
      END PROGRAM temperature
```

Read Loop Continued

Consider the following completed version of the temperature program:

```
PROGRAM temperature
    IMPLICIT NONE
    REAL :: cvalue, fvalue
    INTEGER :: ios
   OPEN (9, FILE = 'fahr.dat')
   OPEN(10, FILE = 'cels.dat')
    READ (9, 100, IOSTAT = ios) fvalue ! Prime read
100 FORMAT(F8.2)
    DO
        IF (ios < 0) EXIT     ! Exit at end of file
       cvalue = (5.0 / 9.0) * (fvalue - 32.0)
       WRITE (10, 200) fvalue, cvalue
 200   FORMAT ('0','Fahr = ',F8.2,TR3,'Cel = ',F8.2)
        READ (9,100,IOSTAT = ios) fvalue
    END DO
   CLOSE( 9)
   CLOSE(10)
   STOP
   END PROGRAM temperature
```

Input

```
85.2
73.0
42.1
32.0
10.0
 0.0
```

Output

```
0Fahr =    85.20   Cel =    29.56
0Fahr =    73.00   Cel =    22.78
0Fahr =    42.10   Cel =     5.61
0Fahr =    32.00   Cel =     0.00
0Fahr =    10.00   Cel =   -12.22
0Fahr =     0.00   Cel =   -17.78
```

Note how the carriage control codes appear in the output file. In an older environment, when the file was sent to the printer the control codes were interpreted as described earlier in these notes. In the modern file-oriented environment, the carriage control codes are merely annoying clutter. As we will see, there are ways to achieve the same results directly (without using carriage control codes at all).

Built-in Functions

Intrinsic or library functions

syntax: Function Name (argument)

Type	Name	Action
real	**REAL**	takes an integer and converts it into a real

Example:

```
INTEGER :: j = 2, k = 7, i
a = k / j                 a <--- 3.0
a = k / REAL (j)          a <--- 3.5
a = REAL (k / j)          a <--- 3.0
i = k / REAL (j)          i <--- 3
```

int **INT** takes a real and converts it into an integer (truncates)

Example:

```
INTEGER :: j = 4, i
REAL :: b = 6.0
i = b / j * 2             i <--- 3
i = INT (b) / j * 2       i <--- 2
i = INT (b / j) * 2       i <--- 2
i = INT (b / j * 2)       i <--- 3
```

int **MOD** returns the remainder upon doing integer division

Example:

```
INTEGER :: j, i
j = 17 / 3                j <-- 5
i = MOD ( 17, 3 )         i <-- 2
```

Note: Functions can be referenced (used) anywhere in an arithmetic expression.

Built-in Functions (cont)

Intrinsic or library functions

Type	Name	Action
real	**SQRT**	takes the square root of a number

EXAMPLE:

```
a = SQRT(9.0)
a = SQRT(10.0)
a = SQRT(b)
a = SQRT( SQRT(81.0) )
```

real	**SIN**	takes the trigonometric sine of a number (angle in radian measure)

Example:

```
y = SIN(x)
y = SIN(5)
```

real	**ABS**	takes the absolute value of a number

Example: Assume REAL :: x = 2.0, y

```
y = ABS(-3.2)
y = ABS(x - 5)
```

or, for integer result (Assume INTEGER:: j) :

```
j = IABS (-3)
```

A summary of these functions and others is given in Appendix A (pages 695-722) of Ellis, Phillips,& Lahey.

Minimal Style & Documentation Guidelines

PROGRAM HEADER: (Required clauses)

- title
- programmer
- due date
- purpose
- ALL variables must be declared, listed and described

VARIABLES:

- Variable names MUST be descriptive.
- Avoid:
  - variables declared but not used
  - variables used but not explicitly declared (i.e., DECLARE ALL VARIABLES)
  - initialization of variables that don't need it (usually sums, counters, and max/mins are initialized)
  - non-initialization of variables that need it
  - 1 letter variable names (only exception: array subscript variables (i, j, etc.))
  - 2 letter variable names (only exception: variable ID)
  - bad variable names (example: IDNUM for an average or OAVG as a sum as in OAVG = OAVG + AVG)

Program Style & Doc (cont)

Misc Considerations

must use good spacing (blank lines) down the page.

no arrays are allowed in programs 1, 2 and 3

must have a explanatory comment before EVERY major block of code, including all:

loops, if-then-elses, summary of results, etc

must have indentation on ALL loops.

must have indentation on all IF-THEN-ELSE statements (words IF, ELSE and ENDIF are not indented). Good consistent indentation!

Note:

Failure to follow these guidelines may result in denial of program consulting by the GTAs or the instructor, due to the increased understanding time that will be incurred.

Style & Doc Example

```
! This program is general guide of a well-documented program.
! Notice the comments, style, indentation, etc.
!
!*************************************************************
!                                                            *
! TITLE   :Triangle Classifier                               *
!                                                            *
! PROGRAMMER:Mark Richard Lattanzi                           *
!                                                            *
! DUE DATE:February 29, 1989                                 *
!                                                            *
! PURPOSE:This program reads in lines of data containing three*
!         integers in I2 format.These numbers represent the  *
!         three sides of a triangle.The program determines   *
!         if the numbers can be made into a triangle, & if so,*
!         what type of triangle they form ( Equilateral,     *
!         Scalene, or Isosceles).                            *
!                                                            *
! VARIABLES:
!         side1 - Integer holding length of first side       *
!         side2 - Integer holding length of second side      *
!         side3 - Integer holding length of third side       *
!         count - Integer holding total number of triangles  *
!         ok    - Integer holding 0 if no triangle can be made*
!         type  - Integer holding the type of triangle       *
!                                                            *
!*************************************************************
```

Style & Doc Ex (cont)

```
! Following 2 lines are not required; they show the col #s
!000000001111111111222222222233333333334444444445555555556666
!234567890123456789012345678901234567890123456789012345678901234567890123

PROGRAM triangle
     IMPLICIT NONE
     ! Declare the types of all variables

     INTEGER :: side1, side2, side3, ok, type, count, ios

     ! Open the disk files needed for disk input/output

     OPEN ( 9, FILE='DEMO.DAT' )
     OPEN (10, FILE='DEMO.RST' )

     ! Initialize all variables that need to be
     ! (sums, max/mins, counters, etc.)
     ! Init counter that holds the # of data lines processed.
     count = 0

     ! Write out the header for the output

     WRITE (10, 501)
     WRITE (10, 502)
```

Style & Doc Ex (cont)

```
      ! Read loop used to read in the lines of data until
      ! the end of file is encountered.

      READ (9, 60, IOSTAT=ios) side1, side2, side3
60    FORMAT (I2,I3,I3)
      DO
           IF (ios < 0) EXIT    ! Exit the loop at end of file
           ! We must count the lines that are processed.
           ! count does not represent the # of valid triangles

           count = count + 1

           ! Determine if the given sides can form a triangle *

           IF (((side1 + side2) .LT. side3) .OR.   &
              ((side1 + side3) .LT. side2) .OR.    &
              ((side2 + side3) .LT. side1)) THEN
              ok = 0
           ELSE
              ok = 1
           END IF
```

Style & Doc Ex (cont)

```
! Triangle type determination for valid triangles.
! All sides equal = equilateral = type 1,
! Nosides equal = scalene= type 2,
! Two sides equal = isosceles= type 3.

IF (ok.EQ.1) THEN
     IF ((side1 .EQ. side2) .AND.       &
        (side2 .EQ. side3)) THEN
        type = 1
     ELSE IF ((side1 .NE. side2) .AND.  &
        (side2 .NE. side3) .AND.        &
        (side1 .NE. side3)) THEN
        type = 2
     ELSE
        type = 3
     END IF
END IF
! Output sides & type or that it is not valid

IF (ok .EQ. 0) THEN
   WRITE (10, 601) side1, side2, side3
ELSE IF (type .EQ. 1) THEN
   WRITE (10, 602) side1, side2, side3
ELSE IF (type .EQ. 2) THEN
   WRITE (10, 603) side1, side2, side3
ELSE
   WRITE(10, 604) side1, side2, side3
END IF
```

Style & Doc Ex (cont)

```
          READ (9, 60, IOSTAT=ios) side1, side2, side3
          ! Loop back and process a new line of data
     END DO

     ! Write out the results and the total
     ! number of lines processed.

100   WRITE (10, 701) count
      WRITE (10, 702)

      STOP

     !Format statements for all output lines

501   FORMAT ('1',TR12,'Triangle Classifier')
502   FORMAT (TR1 , TR5,'Determines types of triangles')
601   FORMAT (TR1 ,'Triangle',I3,',',I3,',',I3,' is not a ',  &
                'triangle.')
602   FORMAT (TR1 ,'Triangle',I3,',',I3,',',I3,' is equilateral.')
603   FORMAT (TR1 ,'Triangle',I3,',',I3,',',I3,' is scalene.')
604   FORMAT (TR1 ,'Triangle',I3,',',I3,',',I3,' is isosceles.')
701   FORMAT ('0','The total number of triangles is ',I3)
702   FORMAT (TR1,'-----------------------------------------')

      END PROGRAM triangle
```

Style & Doc Ex (cont)

Input:

```
 3  4  5
10 10 10
10 10 15
10 11 12
10 10 20
10 10 21
```

Output:

```
1            Triangle Classifier
       Determines types of triangles
 Triangle  3,  4,  5 is scalene.
 Triangle 10, 10, 10 is equilateral.
 Triangle 10, 10, 15 is isosceles.
 Triangle 10, 11, 12 is scalene.
 Triangle 10, 10, 20 is isosceles.
 Triangle 10, 10, 21 is not a triangle.
0The total number of triangles is   6
 ----------------------------------------
```

The program triangle produces the output shown above from the given input file. (Actually, there's one slight difference, but it's not important.)

Notes:

• The count of triangles is incorrect – as predicted in the program documentation.

• The classification of the fifth "triangle" (sides 10, 10, 20) is probably not what most of us would prefer. How could you modify the program to fix that?

• The first triangle (sides 3, 4, 5) is a right triangle. How could the program be modified to add that classification?

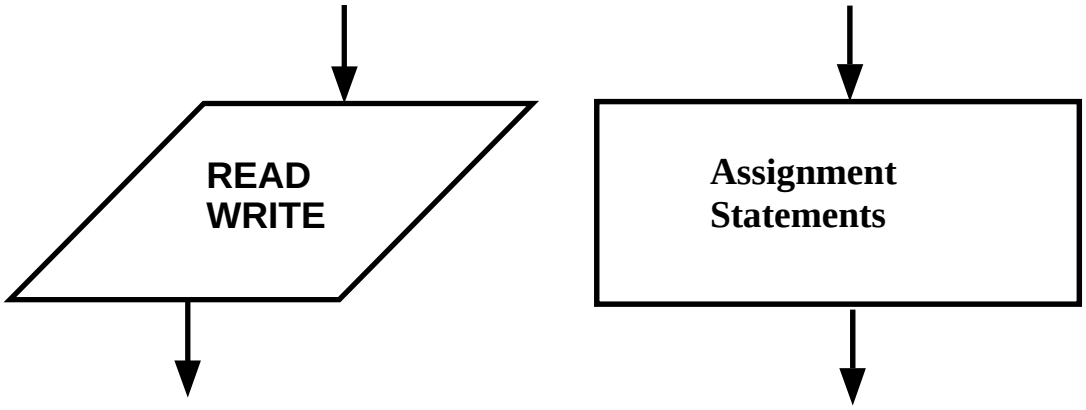
•The carriage control codes produced by the program are shown here exactly as they appear in the output file (no interpretation). What would the output look like if the ccc's were interpreted? How could you modify the program to produce that result directly?

Flow Charts (Execution Flow Diagrams)

- graphical representation of a program

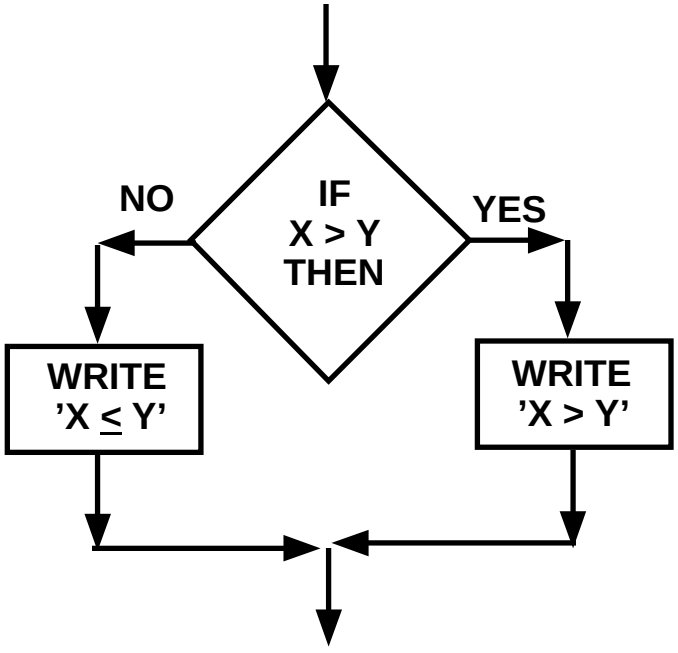
1) sequence of actions - perform one action after another

READ(9, 100) a, b WRITE (10, 200) a, b c = a + bWRITE (10, 300) c



2) decision making (alternation) - decide which of two paths to take

IF THEN ELSE statements



Flow Charts (Execution Flow Diagrams)

3) looping (repetition) - perform the same action over and over

DO… END DO statement